

MARIN VLADA
IOAN NISTOR

ADRIAN POSEA
CĂLIN CONSTANTINESCU

GRAFICĂ PE CALCULATOR ÎN LIMBAJELE PASCAL ȘI C

Implementare și aplicații



volumul I: IMPLEMENTARE



EDITURA

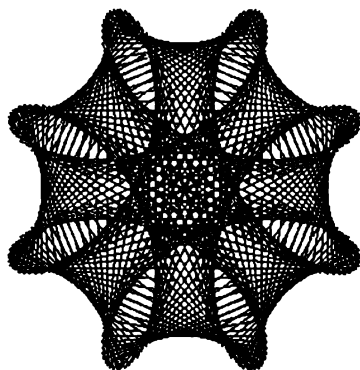
MARIN VLADA
IOAN NISTOR

ADRIAN POSEA
CĂLIN CONSTANTINESCU

GRAFICĂ PE CALCULATOR ÎN LIMBAJELE PASCAL ȘI C

Implementare și aplicații

volumul I: IMPLEMENTARE



EDITURA TEHNICA,
BUCUREȘTI 1992

Copyright © 1991, Editura Tehnică
Toate drepturile asupra acestei ediții
sînt rezervate editurii

Adresa: Editura Tehnică
Piața Presei Libere, 1
33 București, România
cod 79738

Control științific:
conf. dr. HORIA GEORGESCU,
conf. dr. OCTAVIAN BÂSCĂ

Redactor:
ing. SILVIA CÂNDEA

Coperta:
ADRIAN AVRAM

Tehnoredactare computerizată:
mat. GHEORGHE SĂVULESCU

Bun de tipar: 25.09.1991
Coli de tipar: 16,25
C.Z.: 76:681,3

IMPRIMERIA "ARDEALUL" CLUJ



ISBN 973-31-0406-X
ISBN 973-31-0408-6

CUVÎNT ÎNAINTE

Prezenta lucrare este o lucrare deosebit de valoroasă și utilă ce prezintă un domeniu de mare actualitate atât din punct de vedere practic, cât și teoretic. Pînă în prezent nu a văzut lumina tiparului nici o carte privind limbajele moderne de programare ca PASCAL și C pentru microcalculatoare, limbaje care oferă multe posibilități grafice.

Aceasta se datorează în mare măsură dificultății domeniului abordat, care pe lîngă cunoașterea unor sisteme de operare și limbaje evolute de programare, presupune și o stăpînire a noțiunilor de geometrie ce stau la baza lucrului pe calculator în mod grafic. În plus, documentația privind subiectul abordat nu este sistematizată și pe alocuri nici completă.

Lucrarea de față vine să umple acest gol, fiind de aceea foarte oportună. În afară de acest aspect, trebuie remarcată rigoarea științifică ce caracterizează materialul, precum și o foarte bună sistematizare a noțiunilor introduse, ceea ce o face atrăgătoare pentru toți cei ce doresc să se inițieze sau să se perfecționeze în grafică pe calculator. Lucrarea poartă amprenta experienței didactice a autorilor.

Lucrarea cuprinde foarte multe exemple de diferite grade de dificultate care vin să ilustreze în mod fericit noțiunile prezentate. Lucrarea este cu atât mai utilă cu cît în noua programă la secția de informatică este prevăzut un curs de „Grafică pe calculator” pentru care acest material este un excelent suport.

Faptul că în lucrare sînt abordate atât particularitățile de implementare pe minicalculatoare cît și pe microcalculatoare, face ca cercul celor interesați să fie foarte larg, depășind cadrul strict universitar. De altfel, această lucrare reprezintă rezultatul colaborării unor specialiști ce lucrează în cercetare, învățămînt și în domeniul elaborării de produse software.

În concluzie, recomand cu caldură apariția acestei lucrări, fiind convins că se va bucura de o foarte bună primire de către cititori.

20 martie 1991

Conf. dr. Horia Georgescu
Universitatea din București
Facultatea de Matematică

PREFAȚĂ

Prezenta carte este destinată studenților, cercetătorilor, cadrelor didactice și tuturor celor ce doresc să acumuleze și/sau să-și perfecționeze cunoștințele în domeniul utilizării minicalculatoarelor și microcalculatoarelor. Cartea este rezultatul colaborării dintre specialiști ce lucrează în învățămînt, cercetare și în domeniul elaborării software-ului (M. VLADA – Universitatea București, A. POSEA – PROXIM S.A., I. NISTOR – INFOTURISM S.A., C. CONSTANTINESCU – Întreprinderea de Calculatoare Electronice S.A. București) la care se adaugă colaborarea unor studenți (SILVIU BRÎNZEI și MARIAN GÎDEA – anul IV, Universitatea București, Facultatea de Matematică). De aici rezultă utilitatea cărții, dar mai ales, valoarea sa științifică și didactică măsurată printre altele de nouitatea și importanța temelor abordate, de problemele și soluțiile analizate din ambele puncte de vedere: matematic și informatic. Desigur, această valoare va fi cunoscută în urma studierii și utilizării cărții de către cei interesați.

MATEMATICA și INFORMATICA sînt științe care utilizează reciproc — una de la cealaltă — metode, tehnici, instrumente și modele pentru investigarea obiectelor proprii, atît la nivel fundamental, cît și aplicativ. Cartea de față utilizează din plin adevărul acestei afirmații.

Lucrarea este o continuare/completare a temelor abordate în cartea „GRAFICA automată în limbajul FORTRAN77 și aplicații” de M. Vlada și A. Posea, Tipografia Universității București, 1990.

Faptul că am ales limbajele PASCAL și C pentru prezentarea temelor de grafică pe calculator, se datorează calităților recunoscute celor două limbaje de programare: limbajul PASCAL a devenit cel mai utilizat limbaj în mediile universitare pentru învățarea informaticii, iar limbajul C a devenit cel mai performant limbaj pentru elaborarea software-ului.

Lucrarea este concepută în două volume, fiecare cu cîte patru capitole. Volumul I abordează aspectele de implementare ale graficii pe calculator, iar volumul II prezintă diverse aplicații ale graficii, unele dintre ele fiind o premieră la noi în țară.

În introducere se prezintă unele aspecte din punct de vedere hardware și software privind performanțele obținute în domeniul aplicațiilor grafice pe

calculatoarele electronice, și în special pe microcalculatoare sub sistemul de operare PC-DOS/MS-DOS.

În capitolul 1, după ce se trec în revistă principalele caracteristici ale limbajului C (DECUS) sub RSX-11M și câteva exemple de programe, se prezintă o bibliotecă de grafică în limbajul C pentru terminalele de tip DAF care oferă facilități grafice pe minicalculatoare, prezentarea ținând seama de următoarele aspecte: definirea unei sesiuni de lucru, trasări de segmente de dreaptă și ștergeri, desenarea textelor, facilitățile modului VT-100, funcții grafice specializate. Capitolul conține exemple de programe scrise în limbajul C pentru utilizarea bibliotecii prezentate, probleme propuse și anexe în care se dau câteva indicații privind operarea și utilizarea bibliotecii.

În capitolul 2, după ce se trec în revistă principalele caracteristici ale limbajului PASCAL (Oregon) sub RSX-11M și câteva exemple de programe, se prezintă o bibliotecă de grafică în limbajul PASCAL ținându-se seama de aspectele enumerate în capitolul 1. Capitolul conține exemple de programe scrise în limbajul PASCAL pentru utilizarea bibliotecii prezentate, probleme propuse și anexe în care se dau câteva indicații privind operarea și utilizarea bibliotecii.

Capitolul 3, după ce prezintă succint sistemul de operare MS-DOS pentru microcalculatoare compatibile IBM-PC și mediul de programare TURBO PASCAL, cuprinde un memorator util pentru folosirea limbajului TURBO PASCAL. Scopul principal al acestui capitol este acela de a cunoaște și utiliza biblioteca grafică GRAPH.TPU care oferă facilități de grafică pe microcalculatoare compatibile IBM-PC. În acest sens, se prezintă un program demonstrativ pentru utilizarea facilităților grafice în TURBO PASCAL, probleme propuse pentru însușirea celor prezentate și o anexă care cuprinde funcții și proceduri apelabile din biblioteca GRAPH.TPU. Ultima parte a capitolului prezintă algoritmi și programe de grafică pe calculator, reprezentarea grafică a diverselor generări geometrice fiind realizată în limbajul TURBO PASCAL.

În capitolul 4, se prezintă probleme practice pentru implementarea interpolării și trasării curbelor plane pe un dispozitiv de afișare grafică de tip rastru. După prezentarea riguroasă a suportului matematic al problemei interpolării, se prezintă detaliat câțiva algoritmi ce implementează interpolarea curbelor plane. Se acordă o atenție specială chestiunilor legate de calculele aritmetice efective necesare pentru implementarea algoritmilor: evitarea depășirilor aritmetice, limitarea erorilor de rotunjire, etc.

În capitolul 5, se prezintă aplicații în teoria deosebit de interesantă a fractalilor. Prezentarea elementelor teoretice cuprinse în acest capitol are ca scop familiarizarea cititorului cu acest domeniu, cunoașterea și înțelegerea aspectelor teoretice de bază, studiul și reprezentarea grafică a mulțimilor JULIA și MANDELBRÖT. Finalizarea rezultatelor obținute în acest domeniu cu concursul studenților menționați, a condus la realizarea unui pachet de

programe scrise în limbajul PASCAL pentru generarea pe calculator a mulțimilor JULIA și MANDELROT. Pentru scoaterea în evidență a complexității temei abordate, sînt prezentate textele sursă ale programelor și procedurilor utilizate, aceasta și pentru a da un exemplu convingător privind proiectarea și elaborarea unei aplicații de programe scrise în limbajul PASCAL. Ultima secțiune prezintă un program în limbajul TURBO-PASCAL pentru reprezentarea structurilor fractale pe microcalculatoarele compatibile IBM-PC. În anexă sînt prezentate cîteva elemente privind utilizarea dispozitivelor grafice pentru minicalculatoare compatibile PDP-11.

În capitolul 6, se prezintă aplicații în geometria „TURTLE”, stilul de grafică fiind — la început — folosit în limbajul LOGO al inteligenței artificiale. Capitolul cuprinde două secțiuni importante: grafica TURTLE în limbajul PASCAL, prezentată prin intermediul unei biblioteci de proceduri scrise în limbajul PASCAL pentru realizarea de aplicații (arbori binari, arbori PERRON, curba KOCH, curba SIERPINSKI, covorul SIERPINSKI, curba HILBERT, curba PEANO, curba DRAGONULUI) și grafică TURTLE în limbajul C, prezentată prin intermediul unei biblioteci de proceduri scrise în limbajul C pentru realizarea de astfel de aplicații.

Capitolul 7 abordează problemele reprezentării grafice pe calculator a curbilor și suprafețelor. După o prezentare succintă a noțiunilor matematice utilizate, se abordează modul concret de implementare în limbajul TURBO PASCAL a reprezentării grafice pentru unele curbe și suprafețe. Deliberat, se scot în evidență unele curbe plane remarcabile (melcul lui Pascal, epirciclopedia, astroida, lemniscata lui Bernoulli, spirala lui Arhimede, etc.), deoarece sînt multe situațiile în care acestea își găsesc aplicații: tehnică, arhitectură, televiziune, medicină și farmacie, creație artistică, etc. Pentru fiecare exemplu descris se prezintă programul corespunzător și imaginile obținute pe calculator.

Ultimul capitol cuprinde algoritmi și metode incrementale de trasare a curbilor de gradul I și II pentru dispozitive periferice de afișare grafică, prezentarea realizîndu-se atît din punct de vedere teoretic, cît și din punctul de vedere al implementatorului. Algoritmii analizați sînt implementați și verificați pe un microcalculator compatibil IBM-PC AT (procesor 80286). Metodele prezentate sînt riguros justificate teoretic, iar procedurile de trasare sînt descrise la un nivel de detaliere apropiat de necesitățile unei implementări hardware. Toate procedurile descrise admit implementări eficiente pe microprocesoarele actuale, cu aritmetică în virgulă fixă, cu sau fără instrucțiuni de înmulțire și împărțire. Programul a fost scris în limbaj de asamblare 80286, folosind o interfață grafică compatibilă EGA/VGA (rezoluția ecranului fiind de 640×350 puncte (EGA) și 640×480 puncte (VGA)). Programul de testare a algoritmilor prezentați este interactiv și utilizează un dispozitiv de intrare de tip „MOUSE” pentru introducerea

parametrilor grafici ai curbelor desenate.

Ținând seama de conținutul cărții, este de datoria noastră să semnalăm lipsa unui capitol care ar fi trebuit să se numească „Grafică în limbajul TURBO C”. Această lipsă este justificată în primul rând din cauza spațiului și, în al doilea rând, din cauza faptului că după ce se acumulează cunoștințe în domeniul graficii în limbajul TURBO PASCAL, nu este prea mare efortul individual pentru învățarea graficii în limbajul TURBO C.

Procedurile și programele din această carte sînt verificate, testate și executate pe calculator.

Pentru lămuriri, calcule și informații suplimentare (recunoaștem că în unele cazuri sînt necesare), recomandăm consultarea bibliografiei de la capitolul respectiv, consultarea unui specialist binevoitor în domeniul respectiv, consultarea autorilor cărții sau, în ultimă instanță luarea deciziei de investigare cu eforturi proprii, care dacă vor fi dublate de imaginație, intuiție, creație, vor conduce în final la satisfacțiile unui adevărat și curajos explorator în domeniile matematicii și informaticii.

Adresăm cuvinte de mulțumire Editurii Tehnice, domnului director dr. ing. Ioan Ganea, redactorului ing. Silvia Cîndea, pentru sprijinul acordat în publicarea acestei cărți, precum și domnilor mat. Gheorghe Răuș și mat. Gheorghe Săvulescu, de la Laboratorul de Birotică din Institutul de Cercetări în Informatică, pentru grija deosebită manifestată în redactarea computerizată a cărții.

AUTORII

CUPRINS GENERAL

Volumul I

| | |
|--|-----|
| INTRODUCERE | 1 |
| 1. GRAFICĂ ÎN LIMBAJUL C (DECUS) | 9 |
| 2. GRAFICĂ ÎN LIMBAJUL PASCAL (OREGON) ... | 61 |
| 3. GRAFICĂ ÎN LIMBAJUL TURBO PASCAL | 87 |
| 4. INTERPOLAREA CURBELOR PLANE | 173 |

Volumul II

| | |
|---|-----|
| 5. APLICAȚII ÎN TEORIA FRACTALILOR | 1 |
| 6. APLICAȚII ÎN GEOMETRIA „TURTLE” | 93 |
| 7. APLICAȚII ÎN TEORIA CURBELOR ȘI SUPRAFEȚELOR | 135 |
| 8. TRASAREA INCREMENTALĂ A CURBELOR DE GRADUL ÎNTÎI ȘI DOI | 169 |

CUPRINS

Volumul I

| | |
|---|-------------|
| CUVÎNT ÎNAINTE | iii |
| PREFAȚĂ | iv |
| CUPRINS GENERAL | viii |
| CUPRINS | ix |
| INTRODUCERE | 1 |
| 1. GRAFICĂ ÎN LIMBAJUL C (DECUS) | 9 |
| 1.1. Utilizarea limbajului C (memento) | 12 |
| 1.2. Exemple de programe scrise în C | 14 |
| 1.3. O bibliotecă de grafică în C | 26 |
| 1.3.1. Definirea sesiunii de desen | 29 |
| 1.3.2. Trasări/Ștergeri | 32 |
| 1.3.3. Desenarea textelor | 34 |
| 1.3.4. Modul VT-100 | 35 |
| 1.3.5. Funcții specializate | 37 |
| 1.3.6. Exemple | 40 |
| 1.3.7. Probleme propuse | 53 |
| 1.3.8. Observații complementare | 54 |
| 1.4. Bibliografie | 59 |
| 2. GRAFICĂ ÎN LIMBAJUL PASCAL (OREGON) | 61 |
| 2.1. Utilizarea limbajului PASCAL (memento) | 65 |
| 2.2. Exemple de programe PASCAL | 65 |

| | |
|--|-----------|
| 2.3. O bibliotecă de grafică în PASCAL | 69 |
| 2.3.1. Definirea sesiunii de desen | 71 |
| 2.3.2. Trasări/Ștergeri | 74 |
| 2.3.3. Desenarea textelor | 75 |
| 2.3.4. Modul VT-100 | 76 |
| 2.3.5. Subrutine specializate | 78 |
| 2.3.6. Exemple | 80 |
| 2.3.7. Probleme propuse | 84 |
| 2.3.8. Observații complementare | 84 |
| 2.4. Bibliografie | 85 |
| 3. GRAFICĂ ÎN LIMBAJUL TURBO PASCAL | 87 |
| 3.1. Sistemul de operare MS-DOS | 89 |
| 3.2. Mediul de programare TURBO PASCAL | 91 |
| 3.2.1. Operarea în TURBO PASCAL | 92 |
| 3.2.2. Memorator pentru utilizarea limbajului TURBO PASCAL | 94 |
| 3.2.2.1. Declarații | 94 |
| 3.2.2.2. Comenzi | 103 |
| 3.2.3. Programe în limbajul TURBO PASCAL | 108 |
| 3.3. Biblioteca grafică TURBO PASCAL | 112 |
| 3.3.1. Prezentare generală | 112 |
| 3.3.2. Definirea sesiunii de desen | 113 |
| 3.3.2.1. Inițializarea unei sesiuni de desen | 113 |
| 3.3.2.2. Detectarea erorilor | 116 |
| 3.3.2.3. Coordonate | 116 |
| 3.3.2.4. Fixare vizor(viewport). Decupare(clipping) | 117 |
| 3.3.2.5. Punct curent | 118 |
| 3.3.2.6. Scalarea și rotația | 119 |
| 3.3.2.7. Fixarea culorii | 120 |
| 3.3.2.8. Închiderea sesiunii de desen | 122 |
| 3.3.3. Desenarea textelor | 122 |
| 3.3.3.1. Setări pentru trasarea caracterelor | 122 |
| 3.3.3.2. Trasarea unui text | 123 |
| 3.3.3.3. Trasări numerice | 123 |
| 3.3.4. Trasări | 124 |
| 3.3.4.1. Trasări de segmente | 124 |
| 3.3.4.2. Trasări de dreptunghiuri, poligoane, cercuri, arce de cerc și elipse | 126 |
| 3.3.4.3. Hașuri | 127 |
| 3.3.5. Rutine la nivel de pixel | 129 |

| | |
|---|------------|
| 3.3.6. Diverse | 130 |
| 3.3.7. Ștergerca și copierea ecranului | 131 |
| 3.3.8. Constante, tipuri de date și variabile definite în biblioteca grafică GRAPH.TPU | 132 |
| 3.3.9. Program demonstrativ — Exemple de utilizare | 136 |
| 3.3.10. Algoritmi și programe de grafică | 150 |
| 3.4. Probleme propuse | 166 |
| 3.5. Anexe | 168 |
| 3.5.1. Lista comenzilor MS-DOS V4.0 | 168 |
| 3.5.2. Cuvinte rezervate în TURBO PASCAL versiunea 5.5 | 170 |
| 3.5.3. Proceduri și funcții în biblioteca grafică GRAPH.TPU | 171 |
| 3.6. Bibliografie | 172 |
| 4. INTERPOLAREA CURBELOR PLANE | 173 |
| 4.1. Interpolarea cu funcții spline cubice | 176 |
| 4.2. Interpretarea fizică a interpolării cu funcții spline .. | 180 |
| 4.3. Considerații de implementare | 181 |
| 4.4. Determinarea analitică a parametrilor curbei | 183 |
| 4.5. Calculul numeric al parametrilor curbei | 186 |
| 4.6. Interpolare cu un număr redus de puncte | 190 |
| 4.7. Trasarea curbelor plane definite parametric | 196 |
| 4.8. Algoritm de trasare eficientă | 201 |
| 4.9. Bibliografie | 209 |

Volumul II

| | |
|---|------------|
| CUPRINS GENERAL | iii |
| CUPRINS | iv |
| 5. APLICAȚII ÎN TEORIA FRACTALILOR | 1 |
| 5.1. Introducere | 3 |
| 5.2. Aproximații de fractali | 6 |

| | |
|---|-----------|
| 5.2.1. Aproximarea mulțimilor compacte | 7 |
| 5.3. Mulțimile JULIA și proprietățile lor | 10 |
| 5.3.1. Proprietăți fundamentale ale mulțimii JULIA | 11 |
| 5.3.2. Metoda BSM (Boundary Scanning Method) pentru reprezentarea grafică a mulțimilor JULIA | 13 |
| 5.4. Mulțimea lui MANDELBROT | 17 |
| 5.4.1. Proprietăți fundamentale ale mulțimii MANDELBROT | 17 |
| 5.4.2. Clasificarea lui SULLIVAN a structurilor de mulțimi JULIA | 19 |
| 5.4.3. Generarea pe calculator a mulțimii MANDELBROT. | 19 |
| 5.5. Mulțimi de secțiune | 20 |
| 5.6. Proceduri și programe grafice | 25 |
| 5.6.1. Desenarea mulțimilor JULIA | 26 |
| 5.6.2. Desenarea mulțimilor MANDELBROT | 29 |
| 5.6.3. Desenarea mulțimilor de secțiune | 31 |
| 5.7. Pachetul de programe FRACTALI | 32 |
| 5.7.1. Prezentare generală | 32 |
| 5.7.2. Procedura de comenzi indirecte FRACTALI.cmd | 35 |
| 5.7.3. Programe sursă | 37 |
| 5.7.4. Biblioteca grafică GRAF.olb | 67 |
| 5.8. Reprezentarea fractalilor în TURBO-PASCAL | 81 |
| 5.9. Anexă | 86 |
| 5.10. Bibliografie | 92 |
| 6. APLICAȚII ÎN GEOMETRIA „TURTLE” | 93 |
| 6.1. Introducere | 95 |
| 6.2. Grafica TURTLE în limbajul PASCAL | 99 |
| 6.2.1. Biblioteca TURTLE.pas | 101 |
| 6.2.2. Aplicații | 106 |
| 6.2.3. Probleme propuse | 126 |
| 6.3. Grafica TURTLE în limbajul C | 127 |
| 6.3.1. Biblioteca TURTLE.c | 127 |
| 6.3.2. Aplicații | 127 |
| 6.3.3. Probleme propuse | 133 |
| 6.4. Bibliografie | 133 |

7. APLICAȚII ÎN TEORIA CURBELOR ȘI SUPRAFEȚELOR .. 135

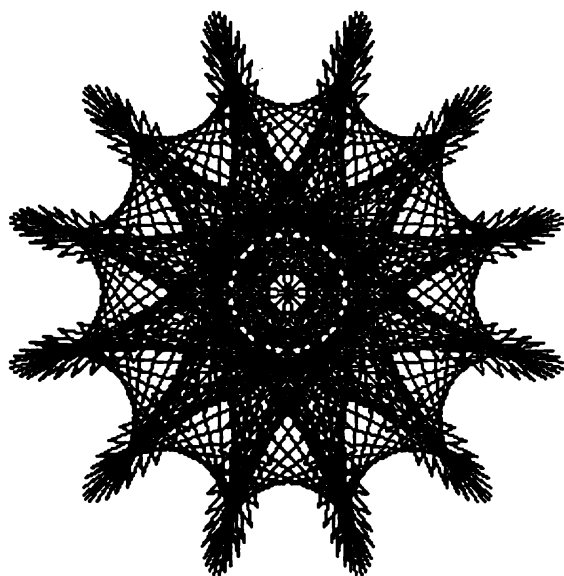
| | |
|--|-----|
| 7.1. Curbe date de ecuații explicite | 138 |
| 7.2. Curbe date de ecuații parametrice | 143 |
| 7.3. Curbe date de ecuații polare | 153 |
| 7.4. Curbe și suprafețe în spațiu | 160 |
| 7.5. Probleme propuse | 167 |
| 7.6. Bibliografie | 168 |

8. TRASAREA INCREMENTALĂ A CURBELOR DE GRADUL ÎNȚÎI ȘI DOI 169

| | |
|--|-----|
| 8.1. Generalități | 172 |
| 8.2. Generarea incrementală a liniilor drepte | 175 |
| 8.2.1. Cazuri speciale, optimizări | 179 |
| 8.3. Generarea incrementală a cercurilor | 180 |
| 8.3.1. Analiza algoritmului de generare în primul octant | 182 |
| 8.3.2. Formule de recurență pentru octanții 2, 3 și 4 | 189 |
| 8.3.3. Chestiuni de frontieră | 192 |
| 8.3.3.1. Traversarea primei bisectoare a axelor | 192 |
| 8.3.3.2. Traversarea axei Oy | 197 |
| 8.3.3.3. Trecerea din octantul 3 în octantul 4 | 198 |
| 8.3.4. Descrierea algoritmului de generare incrementală a cercurilor | 200 |
| 8.3.5. Generarea cercurilor cu coordonatele centrului și raza semiîntregi | 205 |
| 8.3.6. Numărul de puncte generate | 209 |
| 8.4. Generarea incrementală a curbelor de gradul 2 | 213 |
| 8.4.1. Cîteva aspecte teoretice | 213 |
| 8.4.2. Ipoteze de lucru | 219 |
| 8.4.3. Deducerea metodei de generare incrementală | 221 |
| 8.4.4. Relații de recurență. Bucla principală a algoritmului de generare | 224 |
| 8.4.5. Trasarea unui arc de curbă | 231 |
| 8.4.6. Trasarea curbei generale de gradul doi | 235 |
| 8.4.7. Curbe degenerare | 244 |
| 8.4.8. Considerații de implementare | 248 |
| 8.5. Concluzii | 249 |
| 8.6. Bibliografie | 249 |

o

INTRODUCERE



R evoluția din domeniul informaticii a fost posibilă datorită performanțelor obținute din punct de vedere hardware și software, performanțe rezultate din progresele cu totul remarcabile ale microelectronicii. Pentru a scoate în evidență aceste performanțe, vom trece în revistă principalele calculatoare electronice actuale din marea diversitate a calculatoarelor utilizate astăzi în lume. După cum se știe, calculatoarele electronice cel mai frecvent utilizate sînt clasificate în: **minicalculatoare** (multi-user) și **microcalculatoare** (single-user), fiecare cu funcții și performanțe bine precizate.

Minicalculatoarele cele mai cunoscute sînt cele compatibile PDP-11 (sub sistemul de operare RSX-11M; memorie pînă la 4Mo, 1Mo = 1024K, disc magnetic de 40-80 Mo) și de tip VAX (sub sistemul de operare VMS; memorie 16Mo, disc Winchester de 1.2Mo). Configurația unui sistem de calcul coordonat de un astfel de minicalculator cuprinde: terminale alfanumerice și grafice, disc magnetic, bandă magnetică, discuri flexibile, imprimantă matricială, digitizor, plotter. Unul dintre producătorii importanți în acest domeniu este firma DEC (Digital Equipment Corporation). Pînă în prezent s-au realizat pe minicalculatoare produse dintre cele mai complexe în domeniile calculelor tehnico-științifice, cercetării-proiectării asistate de calculator și altele.

Microcalculatoarele din punct de vedere al capacității și al utilizării se clasifică în:

- **familiale** (home computer) cu microprocesor pe 8 biți (Intel 8080, Z80, 6502), memorie dinamică pînă la 48K; exemple: SINCLAIR-ZX81, ZX-SPECTRUM, COMMODORE-16;
- **semiprofesionale** (sub CP/M) cu microprocesor pe 8 biți, memorie dinamică pînă la 64K; exemple: COMMODORE-64, APPLE;
- **profesionale** (sub DOS) cu microprocesor pe 16 sau 32 biți (8086, 8088/XT, 80286/AT, 80386), memorie dinamică pînă la 1Mo; exemple: cele compatibile IBM-PC/XT/AT și noile modele PS/2, PS/1 (Personal System); disc Winchester de 10Mo (XT), 20Mo (AT), disc floppy de 360K (XT), 1.2Mo (AT).
- **portabile** (tip geantă diplomat) avînd afișare cu cristale lichide; se pot cupla la o rețea telefonică; exemple: TRS-100 (Tandy), HP-110, EPSON-Geneva.
- **de capacitate mare**; emulează orice tip de microprocesor, memorie

dinamică peste 1Mo, memorie externă 100Mo, adresare virtuală $2^{30}, \dots, 2^{32}$ octeți; 1 milion de instrucțiuni/secundă; exemple: ALTO-Xerox, DORADO-Xerox, MicroVAX-DEC, SIMBOLICS-3600 (mașină LISP).

Configurația unui sistem de calcul coordonat de un calculator personal din familia IBM-PC poate să cuprindă: plottere, scannere, digitizoare, imprimante grafice. De asemenea, pentru mărirea performanțelor acestor microcalculatoare s-au realizat rețele de microcalculatoare.

Principalii producători de hardware sînt: AT&T, DEC, Bull, IBM, Fujitsu, Olivetti, Philips, Siemens, Sun, Unisys, Gemini Technology Inc., American Megatrends Inc., Xerox, Hewlett Packard, LSI Logic, Bipolar Integrated Technology. Principalii producători de software sînt: Borland International Inc., Lotus Microsoft, Oracle, Palatational Technology. Primii doi mari producători în domeniul tehnicii de calcul și informaticii mondiale, rămîn totuși IBM și DEC.

Un domeniu relativ nou, apărut prin impactul produs de performanțele obținute de calculatoarele personale, este domeniul stațiilor de lucru ingineresti care includ stațiile grafice, echipamente pentru: proiectare asistată de calculator (CAD), fabricația asistată de calculator (CAM), ingineria asistată de calculator (CAE). Firma DEC a intrat pe piața internațională a stațiilor grafice în anul 1984 cu prima stație VAX pentru satisfacerea necesităților impuse de aplicațiile CAD/CAM/CAE.

Odată cu apariția calculatoarelor personale (Personal Computer) s-au produs schimbări și în domeniul graficii pe calculator. Pentru realizarea funcțiilor grafice, pînă atunci, aveau loc o prelucrare de date pe o unitate centrală (memoria calculatorului) și o transmisie a datelor către un terminal grafic. Prin performanțele obținute, acest terminal a evoluat foarte mult ca posibilități, prelufnd la nivel local aproape toate sarcinile graficii. Această evoluție a produs transformarea stațiilor grafice în unități de calcul autonome ce cuprind unul sau mai multe procesoare pentru realizarea facilităților de grafică. O importanță deosebită în acest sens o are memoria video ce este conținută în spațiul propriu de adresare al procesorului. Imaginea este formată în memoria video direct fără a mai fi nevoie de o transmisie, obținîndu-se astfel o mare viteză de execuție. Scrierea direct în memoria ecran este numai o fază a procesorului. Pentru afișarea propriu-zisă această memorie video este citită secvențial de un bloc logic, independent de procesor, care realizează semnalele pentru monitorul TV, conform standardului acestuia. Aceste funcții sînt realizate de adaptorul grafic (graphics adapter) numit și cuplor grafic sau cuplor video. Primele cuploare grafice ce au echipat microcalculatoarele din familia IBM-PC/XT au fost HERCULES și CGA (Color Graphics Adapter). Astăzi, cele mai cunoscute cuploare grafice ce echipează microcalculatoarele din familia IBM-PC/AT sînt EGA (Enhanced Graphics Adapter), apărut în 1984 și VGA (Video Graphics Array), apărut

În 1987. Performanțele unui cuplor grafic sînt măsurate de: viteza de execuție, rezoluția (dimensiunea în pixeli; puncte), număr de culori simultane. Deja, au apărut standarde IBM pentru cuploarele grafice:

| Cuplör | Rezoluție (pixeli) | Număr culori | Memoria Video (Ko) |
|--|--------------------|--------------|--------------------|
| CGA – Color Graphics Adapter | 640×200 | 16 | 16 |
| EGA – Enhanced Graphics Adapter | 640×350 | 16 din 64 | 256 |
| VGA – Video Graphics Array | 640×480 | 16 | 256 |
| VGA – Virtual Graphics Adapter | 640×480 | 16 din 256 | 256 |
| HGC – Hercules Graphics Card (nu aparține IBM) | 720×348 | 2 | 64 |

Pentru aplicațiile grafice, unde fidelitatea imaginii este foarte importantă (procesarea de imagini, simulări de procese complexe — vezi teoria fractalilor — etc.) este nevoie de performanțe deosebite ale cuplorului grafic folosit. Pentru aceste aplicații, deja firma IBM a dezvoltat cuplorul grafic AGA (Advanced Graphics Adapter) cu o rezoluție de 1024×768 pixeli. Recent, cea mai bună rezoluție realizată este de 4096×4096 pixeli.

Dezvoltarea cuploarelor grafice și simultan cu acestea și a dispozitivelor periferice grafice, a fost impulsionată de faptul că prezentarea grafică a rezultatelor aplicațiilor prezintă diverse avantaje: forma atractivă de prezentare, înțelegere mai rapidă, etc. De asemenea, rezultatele obținute în domeniul cercetării-proiectării asistate de calculator, au determinat o evoluție rapidă a produselor software.

PRODUSE HARDWARE PENTRU APLICAȚII GRAFICE

Firmele producătoare de echipamente grafice pentru microcalculatoare, ale căror produse s-au impus ca standarde sînt: IBM, EPSON, TOSHIBA, XEROX, HEWLETT-PACKARD. Echipamentele destinate aplicațiilor grafice sînt următoarele: plottere, digitizoare, scannere, imprimante grafice, mouse.

PLOTTERELE sînt destinate desenării imaginilor realizate de programele de aplicații, desenele executîndu-se cu tuș, cerneală sau pastă

pe un suport de hîrtie. Aceste dispozitive au o precizie mare de execuție și pot executa orice tip de desen în urma prelucrărilor pe calculator. Deoarece pentru forma finală a desenului este nevoie de testări prealabile, execuția desenelor se realizează pe mai multe tipuri de hîrtie. Orice stație grafică destinată activităților de cercetare-proiectare asistată de calculator, trebuie să cuprindă și un astfel de dispozitiv de desen. Performanțele obținute de aceste dispozitive au determinat și dezvoltarea unor produse software corespunzătoare cu mari performanțe. La noi în țară pînă în prezent au fost utilizate plotterele de tip ARISTO (RFG), BENSON (SUA), DIGIGRAF (Cehoslovacia), HEWLETT-PACKARD (SUA) cuplate la minicalculatoare.

DIGITIZOARELE sînt dispozitive de introducere în calculator a coordonatelor numerice ale unor puncte sau curbe de pe un desen deja realizat, coordonate raportate la un sistem rectangular de axe atașat suprafeței de lucru. Această acțiune are loc în scopul reproducerii desenului, în urma unor modificări, completări sau alte prelucrări, pe un plotter sau pe o imprimantă grafică. Primele dispozitive de acest fel au fost utilizate în geodezie și cartografie fiind cunoscute sub numele de coordonatografe. Aplicațiile digitizoarelor în sistemele grafice interactive sînt foarte variate. De exemplu, sînt utilizate în proiectarea asistată de calculator în geodezie, cartografie, mecanică, electrotehnică, arhitectură, construcții etc., atît pentru introducerea coordonatelor și dimensiunilor de pe desene, schițe și planuri, cît și ca dispozitive de interacțiune în conjuncție cu un terminal grafic. Performanțele acestor dispozitive sînt date de numărul de funcții predefinite oferite de către driver (program livrat împreună cu echipamentul) precum și de rezoluție. Cele mai performante digitizoare au un număr de 100-150 funcții predefinite, iar rezoluția pînă la 100 puncte distincte de digitizare pe centimetru.

SCANNERE-le sînt dispozitive destinate prelucrării imaginilor de pe fotografii prin codificare numerică și introducerea lor în fișiere ce ulterior vor fi prelucrate cu diverse programe (editoare grafice, pachete de desktop publishing). În urma prelucrărilor, imaginile obținute pot fi vizualizate pe monitor sau pe imprimanta grafică. În general, scannere-le au o rezoluție de 300 dpi (puncte/inch), diferența calitativă dintre ele fiind dată de viteza de parcurgere a imaginilor. De exemplu, scanner-ul PAGE READER model 245 parcurge o pagină de 27×20 cm în 30 de secunde.

IMPRIMANTELE GRAFICE sînt dispozitive destinate preluării imaginilor de pe monitorul calculatorului. Rezoluția acestor dispozitive este măsurată în dpi, 180 dpi fiind o rezoluție bună pentru aceste dispozitive. Pînă în prezent au apărut mai multe tipuri de astfel de dispozitive cu performanțe din ce în ce mai mari, ajungîndu-se la imprimantele cu laser cu rezoluții de 300 dpi și chiar de 600 dpi, acestea putînd fi comparate cu cele mai moderne echipamente de tipărit.

Dispozitivul de intrare „MOUSE”

Utilizarea dispozitivului de intrare MOUSE pe calculatoarele personale prin intermediul unor produse ca: MICROSOFT WINDOWS, PC PAINTBRUSH, XEROX VENTURA PUBLISHER, etc., a determinat un interes deosebit din partea utilizatorilor. Sistemul de operare gestionează acest dispozitiv prin intermediul unui program specializat (driver). Acesta interacționează cu calculatorul pentru desenarea pe ecran a cursorului și pentru a controla mișcările acestuia. De regulă, prezența unei săgeți indică existența dispozitivului mouse. Software-ul de mouse este implementat doar pe anumite moduri video (modul ecran) dat de tipul adaptorului grafic și anume: C = CGA, E = EGA, M = MDA, H = Hercules, P = PC3270. Mouse-ul vede ecranul fizic al calculatorului ca o matrice de puncte ce reprezintă ecranul virtual și utilizează perechi de puncte virtuale (X, Y) pentru localizarea oricărui punct sau obiect de pe ecran. Există o corespondență de 1:1 între punctele din ecranul virtual și pixelii din ecranul fizic. Mouse-ul există sub formă de **cursor grafic**, care este o matrice de pixeli (16×16) ce se deplasează peste imaginile de pe ecran, sub formă de **cursor text software**, care este un atribut de text și care poate fi deplasat de la un caracter la altul, și sub formă de **cursor text hardware**, care poate fi obținut prin modificarea formei cursorului implicit dat de adaptorul grafic respectiv. La un moment dat poate fi activ doar un singur cursor. Driver-ul de mouse realizează funcții privind selectarea tipului de cursor dorit, modificarea atributelor cursorului, salvarea și restaurarea unor porțiuni de pe ecran, controlează mișcările cursorului exprimate prin mărime, direcție și durată.

PRODUSE SOFTWARE PENTRU APLICAȚII GRAFICE

Aplicațiile grafice se realizează prin utilizarea următoarelor clase de produse software: editoare grafice, biblioteci grafice, programe grafice specializate, desktop publishing, worksheet graphics (birotică).

EDITOARELE GRAFICE sînt destinate prelucrării grafice în domeniul proiectării asistate de calculator și au următoarele funcții importante:

- mărirea sau micșorarea desenelor (ZOOM); editarea de simboluri;
- trasarea în diverse sisteme de coordonate (2D și 3D) și tipuri de proiecții;
- selectarea sistemului de măsură și a dimensiunii paginii de desen;
- localizarea obiectelor prin referință la alte obiecte;
- selectarea culorilor și a tipurilor de linie;
- scalări și rotații ale obiectelor selectate;
- hașuri diverse pentru poligoane;
- utilizarea textelor cu peste 60 tipuri de caractere în alfabetele LATIN, CHIRILIC, GREC;

- utilizarea de simboluri matematice, astronomice, meteorologice, muzicale, etc. cu facilități de dispunere a acestora pe suprafața de desen; cotarea automată a obiectelor;
- atribute de vizibilitate, prioritate, culoare pentru obiecte și posibilitatea editării folosind aceste atribute;
- introducerea de noi elemente grafice cu ajutorul unor tablete grafice sau mouse („șoricelul”);
- salvarea imaginilor în fișiere;
- trasări de curbe, suprafețe și linii poligonale;
- facilități de trasare cu realizarea impresiei de 3 dimensiuni.

Dintre produsele care înglobează editoare grafice amintim: **AUTO CAD, FREELANCE 2 PLUS, WINDOWS PAINT, PC PAINTBRUSH, DELUXE PAINT, MINI CAD 2, GENERIC CAD.**

BIBLIOTECILE GRAFICE sînt destinate prelucrărilor grafice prin intermediul limbajelor de programare de nivel înalt (**FORTRAN, PASCAL, C** etc.). Aceste biblioteci conțin rutine (primitive) grafice care realizează funcțiile grafice necesare aplicațiilor grafice. Pentru minicalculatoare (sub **RSX-11M**) există astfel de biblioteci sub diverse forme, performanțele lor au dus la realizarea de aplicații grafice complexe. Pentru microcalculatoare (sub **MS-DOS**) reprezentative sînt bibliotecile compilatoarelor **TURBO PASCAL** și **TURBO C**. Ținînd seama de portabilitatea limbajului **C**, există biblioteci independente ce conțin rutine apelabile din mai multe limbaje de programare de nivel înalt. Aceste rutine sînt scrise în limbajul **C**. În general, o bibliotecă grafică trebuie să conțină următoarele:

- rutine de inițializare a sesiunii de grafică; acestea selectează modul grafic și stabilesc zonele de memorie pentru scrierea și afișarea imaginilor;
- rutine pentru stabilirea zonei coordonatelor vizibile ale desenului (spațiul utilizator) și zona activă de pe ecran (spațiul dispozitiv);
- rutine pentru selectarea culorilor;
- rutine pentru stabilirea atributelor: culoare de trasare, stil de linie, grosime linie etc.;
- rutine pentru trasarea liniilor, arcelor, elipselor, poliliniilor, trasarea hașurilor etc.;
- rutine pentru afișarea textului;
- rutine pentru copierea imaginii grafice la imprimantă;
- rutine pentru gestiunea memoriei ecran.

PROGRAMELE GRAFICE SPECIALIZATE sînt destinate rezolvării problemelor din anumite domenii, oferind utilizatorului posibilitatea să enunțe problemele cît mai simplu și în concordanță cu limbajul utilizat în domeniul său. În general, aceste programe cuprind două părți: nucleul, ce efectuează calculele sau operațiile corespunzătoare domeniului respectiv (exemple: calcule matematice, vezi **MATHEMATICA** și **MATHCAD**; calcule tehnico-științifice,

vezi **EUREKA**), și interfața cu utilizatorul, ce transferă informația de la utilizator la calculator și asigură prezentarea și procesarea grafică a rezultatelor. Nucleul unui astfel de program de grafică specializat este scris în: **C. MATHEMATICA** este un produs program care realizează calcule matematice în mod simbolic, adică operează în mod simbolic cu ecuații și formule, face reduceri și descompuneri în factori, derivează și integrează funcții (în cazul în care pentru integrală există soluția analitică; dacă integrarea simbolică nu este posibilă, atunci programul prezintă o soluție numerică). Reprezentarea rezultatelor diverselor probleme se poate realiza în două sau trei dimensiuni, în coordonate rectangulare sau sferice. Funcțiile grafice ale programului oferă posibilități pentru procesarea textelor matematice, fiind un foarte bun editor de texte matematice. **MATHCAD** este un alt program de grafică specializat cu funcții asemănătoare cu **MATHEMATICA**. **EUREKA** este un produs program care realizează calcule tehnico-științifice prin tehnici numerice, dar cu posibilitatea unor calcule simbolice tip **PROLOG** sau **LISP**. Introducerea datelor se face folosind limbajul matematic uzual. Problema ce trebuie rezolvată se prezintă prin ecuații, sisteme, relații etc., care se declară prin notații matematice uzuale. Programul oferă posibilități de documentare pentru declararea problemelor, rezultatele putând fi prezentate sub formă grafică.

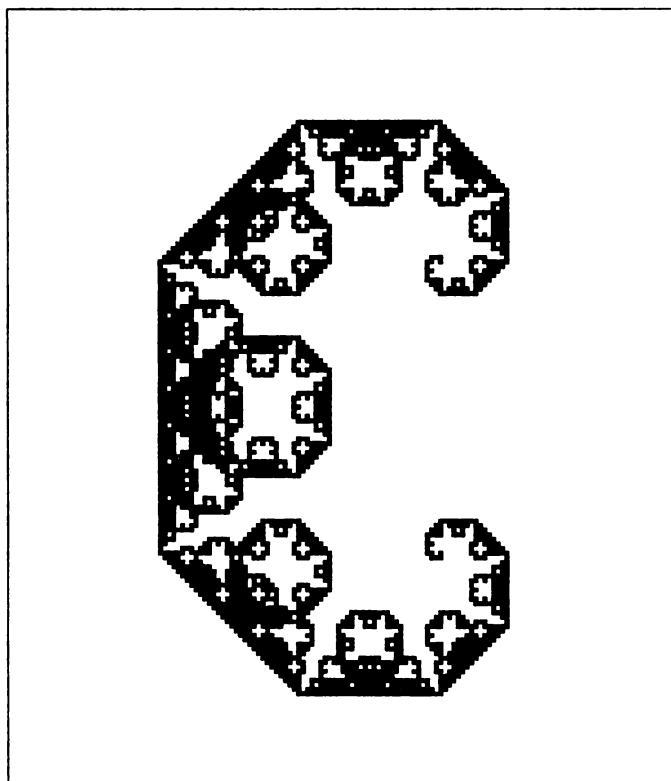
DESKTOP PUBLISHING sînt produse software destinate realizării de publicații (ziare, reviste, reclame etc.) la birou, parcurgînd toate etapele de la o tipografie clasică. Principalele funcții ale acestor produse sînt:

- redactarea documentului cu ajutorul unui procesor de texte;
- editarea și corectarea documentului;
- includerea textului în pagină;
- inserarea în text a desenelor realizate automat sau preluate prin intermediul unui scanner;
- realizarea design-ului paginii;
- machetarea paginilor și tipărirea conform machetei.

Un produs performant și foarte utilizat este **XEROX VENTURA PUBLISHER** ce este compatibil cu mai multe procesoare de text: **WORDSTAR**, **WORD PERFECT**, **MULTIMATE** și unele editoare grafice: **AUTOCAD**, **PAINTBRUSH**, **WINDOWS PAINT**. Alte produse cu aceste funcții sînt: **PAGEMAKER**, **XPress**, **SUPER PAINT**, **PUBLISH IT**.

WORKSHEET GRAPHICS (Birotică) sînt produse software destinate aplicațiilor de evidență din domeniul financiar-contabil care pot realiza: analiza vânzărilor, tabele de cîștiguri și pierderi, salarii, date personale ale salariaților, etc. Datele sînt introduse în celule care se află în cadrul unor tabele. Grafica este folosită la prezentarea ieșirilor sub formă de diagrame, grafice, tabele etc. Cel mai cunoscut produs cu aceste funcții este **LOTUS** (realizat de firma **LOTUS**). Alt produs este **QUATTRO** realizat de **BORLAND INTERNATIONAL**.

GRAFICĂ ÎN LIMBAJUL C (DECUS)



| | |
|---|----------|
| 1. GRAFICĂ ÎN LIMBAJUL C (DECUS) | 9 |
| 1.1. Utilizarea limbajului C (memento) | 12 |
| 1.2. Exemple de programe scrise în C | 14 |
| 1.3. O bibliotecă de grafică în C | 26 |
| 1.3.1. Definirea sesiunii de desen | 29 |
| 1.3.2. Trasări/Ștergeri | 32 |
| 1.3.3. Desenarea textelor | 34 |
| 1.3.4. Modul VT-100 | 35 |
| 1.3.5. Funcții specializate | 37 |
| 1.3.6. Exemple | 40 |
| 1.3.7. Probleme propuse | 53 |
| 1.3.8. Observații complementare | 54 |
| 1.4. Bibliografie | 59 |

In 1978 B. Kernighan și D. Ritchie au publicat cartea „*The C programming language*”, receditată în 1988. În Introducere se precizează: „Multe din ideile mai importante din C provin din bătrînul dar încă vitalul BCPL, dezvoltat de Martin Richards. Influența lui BCPL asupra lui C s-a făcut indirect prin intermediul limbajului B, conceput de Ken Thompson în 1970 pentru primul sistem de operare UNIX de pe calculatoarele PDP-7”. Între timp au mai apărut multe cărți de acest fel, bine primite de publicul interesat. În 1986 s-a propus standardizarea limbajului C după zece ani de experiență a creatorilor compilatorului și de entuziasm al utilizatorilor lui.

Rezultatele remarcabile se datorează desigur caracteristicilor lui, în primul rînd fiind aceea de a fi limbaj pentru profesioniști, adică pentru cei care realizează produse-program ca sarcină de serviciu, disciplinat, conștienți de realitățile hardware pe care limbajul nu le ascunde ci le cere frecvent utilizate. De aceea codul generat are calități pe care codul generat de alte compilatoare nu le are, fiind pe bună dreptate numit asamblor independent de mașină. Limbajul C este apropiat de un limbaj de asamblare, dar lucrează cu structuri de date și de control pe care nici limbajele de nivel înalt nu le posedă. Limbajul C este „mic”, ușor de învățat, astfel că este mai ușor de implementat compilatorul C pe toată gama de echipamente decît orice alt compilator. De aici rezultă portabilitatea lui deosebită. Compilatorul C acceptă surse considerate criptice; nu face verificări pe care alte compilatoare (Pascal) le fac, nu are definite funcții intrinseci pentru calcule matematice, pentru prelucrări de șiruri de caractere, pentru operații de I/O, etc. Toate aceste funcții există (sau pot fi scrise de utilizator) separat și se înglobează în task-ul final la editarea de legături. Această lipsă inițială se datorează dorinței de a fi independent de aplicație, nesubordonat nici unei „filozofii”, dar a impus și prin aceasta existența unui mediu de programare structurată. Codul obiect generat este mai slab decît unul produs de un asamblor, dar compilatorul lucrează în doi pași, în al doilea acceptînd un text realizat în prima fază, eventual ajustat cu un editor (inserare de cod în limbaj de asamblare).

Un program scris în limbajul C se preprocesează (avînd directive de

includere fișiere, de macrosubstituție, de compilare condiționată) apoi se prelucrează textul rezultat care conține:

- definirea și utilizarea variabilelor de tip pointer;
- declarații concise pentru o mare varietate de date (structurate);
- instrucțiuni de control și operatori aritmetici care exclud artificiiile de programare prezente în alte limbaje;
- definiri și referiri de date sau funcții locale sau globale (există loc pentru mai bine, dar oferta actuală nu limitează ci doar creează un oarecare disconfort pentru programator);
- funcții apelate întotdeauna prin valoare (care poate fi adresa unui obiect pe care l-am dori accesat prin adresă); este permis apelul recursiv.

Mai multe firme de software au scris compilatoare de C: Microsoft, Borland, DEC, etc. Probabil C nu va mai fi la fel de portabil ca în prezent (cum s-a întâmplat cu limbaje standardizate ca FORTRAN, BASIC, etc.). Dar contribuția pe care a adus-o la definirea unui limbaj al programatorului de sistem (de operare) rămâne fundamentală.

1.1. Utilizarea limbajului C (memento)

Alfabetul limbajului este compus din litere (A-Z, a-z), cifre (0-9), caractere speciale tipăribile (+ - * / % () _ ~ & | \$ # ! ? < > . , ; " ' { } \ [] ^ =) și netipăribile (NUL, BS, HT, NL, FF, CR). Se ignoră spațiile albe definite de caracterele spațiu, tab, linie vidă, comentarii (introduse prin /* comentariu */).

Cuvintele cheie ale limbajului sînt: auto, break, case, char, continue, default, do, double, else, extern, float, for, goto, if, long, register, return, short, sizeof, static, struct, switch, typedef, union, unsigned, while.

Tipurile de date intrinseci sînt: char (-128,127), int (-32768, 32767) unsigned int (0,65535), long int (-2^{31} , $2^{31}-1$), float (3.4E-38, 3.4E+38), double (1.7E-308, 1.7E+308). Constantele caracter se introduc prin 'c' sau '\xxx' unde xxx poate fi a(007), b(010), f(014), n(012), r(015), t(011), v(013), \ (0134), "(042), ?(075) sau orice combinație de trei cifre octale. Constantele întregi cu primul caracter 0 se consideră octale, cele cu primele două caractere 0x se consideră hexazecimale. Se pot utiliza sufixele L (long) și U (unsigned).

Variabilele sînt principalele date manipulate de un program. Ele sînt caracterizate prin tipul lor (indicînd mărimea memoriei ocupate) și clasa de memorie folosită (static, extern, auto, register). Toate variabilele se declară (eventual cu inițializare) prin construcții de forma

[clasa] tip nume[=valoare-initala];
unde [și] delimitează o construcție opțională. Se face distincție între literele mari și mici în numele de variabilă.

Datele structurate sînt fie liste indexate (începînd de la 0) de componente de tipul de bază al listei (tabloului) sau sînt definite de utilizator.

Operatorii aritmetici (+, -, *, /) sînt folosiți cu semnificația cunoscută din alte limbaje. În plus, C oferă următorii operatori:

| | |
|----------|---|
| [] | indexare |
| . | selectare |
| -> | dereferențiere și selectare |
| * | indirectarea dereferențierii |
| ++ | incrementare |
| a++, ++a | sînt echivalente (cu nuanțe) cu a=a+1 |
| -- | decrementare |
| a--, --a | sînt echivalente (cu nuanțe) cu a=a-1 |
| && | 'și' logic |
| | 'sau' logic |
| == | 'egal cu' logic |
| != | 'diferit de' logic |
| = | atribuire |
| op= | atribuire rapidă |
| a op= b | este echivalent (cu nuanțe) cu a = a op b |
| &, , ^ | și, sau, sau exclusiv (pe biți) |

Programele sînt alcătuite dintr-o colecție de fișiere sursă conținînd descrieri de date și funcții. În unul dintre ele trebuie să existe și main() sau main(n, l), unde n este numărul argumentelor cu care se lansează programul, iar l este adresa listei celor n șiruri de caractere prezente. Fiecare fișier se compilează individual și eventual codul obiect generat se depune într-o bibliotecă.

Directivele preprocesorului sînt:

| | | |
|-------------------|----------------------------------|----|
| #include <fisier> | /* din LB:[1,1] | */ |
| #include "fisier" | /* din SY: | */ |
| #define ident sir | /* macrosubstitutie | */ |
| #undef ident | /* anuleaza #define ident | */ |
| #ifdef ident | /* compilare conditionata de | */ |
| #ifndef ident | /* (ne)definirea lui ident | */ |
| #if expresie | /* sau daca expresie != 0 | */ |
| #else | | |
| #endif | /* sfirsit bloc compilare cond*/ | |

Compilarea se face (sub RSX-11M trebuie ca cc să fie instalat sub numele xcc, iar AS sub numele xas) cu comanda xcc -a fișier, unde a este un switch dintre cele permise; lista acestora se află cu xcc -h.

Compilatorul C DECUS definește simbolul `decus` (care poate fi testat cu `#ifdef decus`); deasemenea el recunoaște directivele `dsect` și `psect`.

Pentru alte lămuriri recitiți [6], [7] sau consultați un cunosător accesibil.

1.2. Exemple de programe scrise în C

Exemplul 1:

Programul SUMA

Calculul $\int f(x)dx$ ca limită $\lim_{n \rightarrow \infty} \sum_{k=0}^n q \cdot f(a + k \cdot q)$, unde $q = \frac{b-a}{n}$.

```
main()
{
    double a, b, f(), q, eps;
    double Suma, LastS, abs();
    int n,k;
    eps = 1.E-5; n=10; LastS=0.0; Suma = 1.0;
    while(abs(LastS - Suma) > eps) {
        n = n*3/2; q=(b-a)/n;
        LastS = Suma; Suma = f(a);
        for (k=1; k<n ; k++)
            Suma += f(a+k*q);
        Suma = q*(Suma + f(b));}
    printf("Valoarea integralei este %f",Suma);
    printf("\ngasita dupa sumarea a %d termeni", n);
}
```

Exemplul 2:

Programul IDENT

```
/*
    Program IDENT
    pentru listarea si modificarea articolului <ident>
    dintr-un fisier .OBJ
    Utilizare:
    - pentru listare (pe stdout, eventual redirectat)
      IDENT fisier
```

- pentru modificare <ident>, conform optiunilor
din stdin (eventual redirectat catre un fisier
creat si actualizat cu orice EDITor)

IDENT M fisier

Nota: se creeaza un nou fisier.OBJ

Autor: Adrian POSEA

Data: 15 Mai 1988

*/

```
#include <stdio.h>
main(argc, argv)
int argc; char *argv[];
{
char buf[255], w[30];
int l,i;
FILE *old_obj,*new_obj;
if(! (old_obj=fopen(argv[1],"ru"))){
error("FCS err %o\n",$$ferr);
if(argc == 2)
while((l=fget(buf, 255, old_obj))){
if(buf[0]==1)
for(i=2; i; i+=8) {
r50toa(w,&buf[i],2);
if(buf[i+5]==0)printf("M name : %6.6s",w);
if(buf[i+5]==6)printf(" ident : %6.6s\n",w);
};
}
else {
new_obj=fopen(argv[1],"wun");
while((l=fget(buf, 255, old_obj))){
if(buf[0]==1)
for(i=2; i; i+=8)
if(buf[i+5]==6){
gets(w);
ascr50(6, &w[24], &buf[i]);
buf[0]=1;
};
fput(buf, l, new_obj);
}
}
}
```

După cum se observă ușor, programul este complet dar nu general
valabil. Generalizarea nefiind complicată o lăsăm ca exercițiu pentru

cunoscătorii structurilor de intrare în TKB.

Exemplul 3

Programul SCAMP

```

/*
  Autor:      Adrian POSEA
  Data:       20 Februarie 1988
  Scop:
    Construiește secvența de caractere de comandă
    pentru SCAMP (RCD-9335), folosind un fișier de intrare
    corespunzător.
  Comutatoare:
    1 - încarcă un caracter definit
      de utilizator în forma:
    replacedchar(octal code),nroflines(decimal 1 to 9)
    format
      Exemplu:
          100,9
          T T T T T
            T
            T
            T
            T
            T
            T
          T
          T

    c - translatează secvențe '\abc' în
      caractere ASCII
      Exemplu:
          \033#6 Text elongat pe DAF-2020
*/

/*)BUILD
$(INCLUDE)      = qiottdrv.h
$(RXLIB)        = c/lb
$(LIBS)         = cx/lb
$(TKBOPTIONS)  = {
    task=...sca
    units=3

```

```

maxbuf=260
stack=1125
}
*/

char c[5]    = {1,2,4,8,16};
char me_switch[]="Scamp -- Switch ilegal";
char me_trunc[]="Scamp -- Intrarea trunchiata";
char usrasc[1748]={
27,76,32,
32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,
32,32,32,32,32,32,32,32,63,34,32,32,32,32,32,32,32,32,
32,32,32,32,32,32,39,32,32,32,32,32,39,32,32,32,32,32,
52,32,32,32,63,35,32,32,52,32,32,32,63,35,32,32,52,32,
36,32,42,33,32,32,42,33,53,34,42,33,32,32,42,33,48,32,
35,34,32,33,51,32,40,32,36,32,34,32,33,35,32,32,32,35,
50,33,37,32,40,34,33,32,54,34,32,32,32,33,32,32,48,34,
32,32,32,32,36,32,32,32,34,32,32,32,33,32,32,32,32,32,
32,32,32,32,32,32,60,32,34,33,33,34,32,32,32,32,32,32,
32,32,32,32,32,32,33,34,34,33,60,32,32,32,32,32,32,32,
40,32,34,33,40,32,52,32,32,32,52,32,40,32,34,33,40,32,
40,32,32,32,40,32,32,32,62,33,32,32,40,32,32,32,40,32,
32,32,32,32,32,32,32,42,32,36,32,34,32,32,32,32,32,32,
40,32,32,32,40,32,32,32,40,32,32,32,40,32,32,32,40,32,
32,32,32,32,32,32,32,35,32,32,32,35,32,32,32,32,32,32,
32,32,32,34,32,33,48,32,40,32,36,32,34,32,33,32,32,32,
27,76,48,
60,32,34,33,32,32,33,34,32,32,33,34,32,32,34,33,60,32,
32,32,32,32,34,34,32,32,63,35,32,32,32,34,32,32,32,32,
34,34,32,33,33,34,48,32,33,34,48,32,41,34,32,32,38,34,
34,33,32,32,33,34,32,32,41,34,32,32,41,34,32,32,54,33,
48,32,40,32,52,32,34,32,49,32,32,32,63,35,32,32,48,32,
39,33,32,32,37,34,32,32,37,34,32,32,37,34,32,32,57,33,
48,33,40,32,36,34,32,32,42,34,32,32,41,34,32,32,48,33,
33,32,32,32,33,34,32,33,49,32,40,32,37,32,34,32,33,32,
54,33,32,32,41,34,32,32,41,34,32,32,41,34,32,32,54,33,
38,32,32,32,41,34,32,32,41,33,32,32,49,32,40,32,38,32,
32,32,32,32,32,32,44,35,32,32,44,35,32,32,32,32,32,32,
32,32,32,32,32,32,44,37,32,34,44,33,32,32,32,32,32,32,
32,32,40,32,32,32,52,32,32,32,34,33,32,32,33,34,32,32,
52,32,32,32,52,32,32,32,52,32,32,32,52,32,32,32,52,32,
32,32,33,34,32,32,34,33,32,32,52,32,32,32,40,32,32,32,
34,32,32,32,33,32,32,32,49,34,32,32,41,32,32,32,38,32,

```

27,76,64,
 60,32,34,33,33,34,40,32,53,34,32,32,53,34,34,32,60,34,
 48,35,40,32,52,32,34,32,49,32,34,32,52,32,40,32,48,35,
 33,34,62,33,33,34,32,32,41,34,32,32,41,34,32,32,54,33,
 62,33,32,32,33,34,32,32,33,34,32,32,33,34,32,32,34,33,
 33,34,62,33,33,34,32,32,33,34,32,32,33,34,32,32,62,33,
 63,35,32,32,41,34,32,32,41,34,32,32,41,34,32,32,33,34,
 63,35,32,32,41,32,32,32,41,32,32,32,41,32,32,32,33,32,
 62,33,32,32,33,34,32,32,33,34,32,32,41,34,32,32,58,33,
 63,35,32,32,40,32,32,32,40,32,32,32,40,32,32,32,63,35,
 32,32,32,32,33,34,32,32,63,35,32,32,33,34,32,32,32,32,
 48,33,32,32,32,34,32,32,32,34,32,32,32,34,32,32,63,33,
 63,35,32,32,40,32,32,32,52,32,32,32,34,33,32,32,33,34,
 63,35,32,32,32,34,32,32,32,34,32,32,32,34,32,32,32,34,
 63,35,32,32,34,32,36,32,40,32,36,32,34,32,32,32,63,35,
 63,35,32,32,34,32,36,32,40,32,48,32,32,33,32,32,63,35,
 62,33,32,32,33,34,32,32,33,34,32,32,33,34,32,32,62,33,
 27,76,80,
 63,35,32,32,41,32,32,32,41,32,32,32,41,32,32,32,38,32,
 62,33,32,32,33,34,32,32,49,34,32,32,33,33,32,32,62,34,
 63,35,32,32,41,32,32,32,41,32,32,32,57,32,32,33,38,34,
 38,33,32,32,41,34,32,32,41,34,32,32,41,34,32,32,50,33,
 33,32,32,32,33,32,32,32,63,35,32,32,33,32,32,32,33,32,
 63,33,32,32,32,34,32,32,32,34,32,32,32,34,32,32,63,33,
 39,32,40,32,48,32,32,33,32,34,32,33,48,32,40,32,39,32,
 63,33,32,34,32,33,48,32,40,32,48,32,32,33,32,34,63,33,
 32,32,33,34,34,33,52,32,40,32,52,32,34,33,33,34,32,32,
 33,32,34,32,36,32,40,32,48,35,40,32,36,32,34,32,33,32,
 32,32,32,32,33,34,32,33,49,34,40,32,37,34,34,32,33,34,
 32,32,32,32,63,35,32,32,33,34,32,32,33,34,32,32,32,32,
 32,32,33,32,34,32,36,32,40,32,48,32,32,33,32,34,32,32,
 32,32,32,32,33,34,32,32,33,34,32,32,63,35,32,32,32,32,
 32,32,32,32,36,32,34,32,33,32,34,32,36,32,32,32,32,32,
 32,40,32,32,32,40,32,32,32,40,32,32,32,40,32,32,32,40,
 27,76,96,
 32,32,32,32,33,32,32,32,34,32,32,32,36,32,32,32,32,32,
 32,33,48,32,36,34,48,32,36,34,48,32,36,34,56,33,32,34,
 63,35,32,32,36,34,32,32,36,34,32,32,36,34,56,33,32,32,
 56,33,32,32,36,34,32,32,36,34,32,32,36,34,40,32,32,32,
 56,33,36,34,32,32,36,34,32,32,36,34,32,32,63,35,32,32,
 56,33,32,32,52,34,32,32,52,34,32,32,52,34,40,32,32,32,
 32,32,32,32,36,32,32,32,62,35,33,32,36,32,33,32,32,32,
 32,32,56,32,36,37,32,32,36,37,32,32,36,37,32,32,60,35,

```

63,35,32,32,36,32,32,32,36,32,32,32,36,32,56,35,32,32,
32,32,32,32,36,34,32,32,61,35,32,32,32,34,32,32,32,32,
32,34,32,36,32,32,32,36,61,35,32,32,32,32,32,32,32,32,
63,35,32,32,48,32,32,32,40,33,32,32,36,34,32,32,32,32,
32,32,32,32,33,34,32,32,63,35,32,32,32,34,32,32,32,32,
60,35,32,32,36,32,32,32,56,35,36,32,32,32,36,32,56,35,
36,32,56,35,36,32,32,32,36,32,32,32,36,32,56,35,32,32,
56,33,32,32,36,34,32,32,36,34,32,32,36,34,32,32,56,33,
27,76,112,
60,39,32,32,36,33,32,32,36,33,32,32,36,33,56,32,32,32,
36,32,36,33,32,32,36,33,32,32,36,33,32,32,60,39,32,32,
32,32,60,35,32,32,40,32,32,32,36,32,32,32,36,32,32,32,
32,32,40,32,52,34,32,32,52,34,32,32,52,34,32,33,32,32,
32,32,32,32,36,32,32,32,63,33,32,34,36,32,32,34,32,32,
60,33,32,32,32,34,32,32,32,34,32,32,32,34,60,33,32,34,
36,32,40,32,48,32,32,33,32,34,32,33,48,32,40,32,36,32,
60,32,32,33,32,34,32,33,48,32,32,33,32,34,32,33,60,32,
32,32,36,34,40,33,32,32,48,32,32,32,40,33,36,34,32,32,
32,32,36,32,40,36,48,34,32,33,48,32,40,32,36,32,32,32,
32,32,36,34,32,33,36,34,48,32,36,34,40,32,36,34,32,32,
32,32,32,32,40,32,34,33,61,34,32,32,33,34,32,32,32,32,
32,32,32,32,32,32,32,32,55,35,32,32,32,32,32,32,32,32,
32,32,32,32,33,34,32,32,61,34,34,33,40,32,32,32,32,32,
34,32,33,32,32,32,33,32,32,32,34,32,32,32,34,32,33,32,
32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,
27,68);
main(argc,argv)
int argc;
char *argv[];
{
char irec[255],orec[255];
int i,j,k,l,nr,nl;
int prmlst[6],iosb[2];
char *addr;

if(argc != 2) goto e_switch;
switch(*argv[1]){
default:goto e_switch;
case('L'):
case('l'):goto load;
case('C'):
case('c'):goto code;
}

```

```

}

ARCC3P(u, v, w)

/*
    Procedura de trasare a unui arc de cerc
    care trece prin 3 puncte

    versiunea 2.3 (Adrian POSEA, 01. 09. 1989)
*/
double u[2] /* punctul initial */,
        v[2] /* punctul intermediar */,
        w[2] /* punctul final */;
{
    double d[2], r, s, t, xc, yc;
    double det();

    d[0] = w[0]; d[1] = w[1];          "
    r = det(u, v, w);
    r = r+r;
    if(r >= zero)
    {d[0] = u[0]; d[1] = u[1];}
    movpen(1, d[0], d[1]);
    s = (N(u)-N(v))/r;
    t = (N(v)-N(w))/r;
    xc = s*(v[1]-w[1]) - t*(u[1]-v[1]);
    yc = t*(u[0]-v[0]) - s*(v[0]-w[0]);
    et1:
    s = xc+(d[0]-xc)*cosu-(d[1]-yc)*sinu;
    d[1] = yc+(d[0]-xc)*sinu+(d[1]-yc)*cosu;
    d[0] = s;
    t = det(u, d, w);
    if(t*r >= zero){
        movpen(11, d[0], d[1]);
        goto et1;}
    movpen(1, w[0], w[1]);
}

```

Exemplul 5:

face parte dintr-un task mai complex care MacroProcesează un fișier .C

Programul MP

```

/*
 * Rutine pentru manipularea tabelii de simboluri a MP
 *
 * Organizarea generala a tabelii de simboluri este
 * bazata pe reprezentarea sub forma de lista
 * inlantuita pentru a descreste timpul de cautare.
 * Substructurile sint definite de primul caracter
 * al simbolului (deci sint necesare 54 de
 * substructuri).
 *
 * Daca aceasta implementare creeaza probleme, se
 * poate propune o alta functie de mapare. Daca nici
 * aceasta nu este adecvata, atunci trebuie incercat
 * un mecanism de lucru cu tabela de simboluri
 * complet diferit (mentinindu-se functiile de
 * stergere simboluri).
 */

/*
 * Include definitii pentru constante globale
 */

#include "mpdefs.h"

struct sym *sym_bkts[54]; /* Substruct tabelii */

/*
 * Cautare in tabela de simboluri
 */

struct sym *lookup(symbol)
    register char *symbol;
{
    register struct sym *ptr;

    for ( ptr = sym_bkts[bkt_map(*symbol)];
          ptr != NIL;
          ptr = ptr->next)
        if (lexeq(symbol, ptr->name))

```

```

        break;
    return(ptr);
}

/*
 * Insereaza un simbol in tabela de simboluri
 */

struct sym *sym_enter(symbol, nargs, defptr)
char *symbol, *defptr;
int nargs;
{
    register int bucket;
    register struct sym *p, *tp;

    bucket = bkt_map(*symbol);
    if ((p = lookup(symbol)) != NIL)
        free(p->defptr); /* elib. bufferul vechii def */
    else {
        /*
         * Aloca un nou nod in tabela de simboluri
         */
        p = get_mem(sizeof *sym_bkts[0]);
        p->next = NIL;
        if(sym_bkts[bucket] == NIL)
            sym_bkts[bucket] = p;
        else {
            for (    tp = sym_bkts[bucket];
                    tp->next != NIL;
                    tp = tp->next);
                tp->next = p;
            }
        }
        strcpy(p->name, symbol); /* Copiaza numele */
        p->nargs = nargs;        /* Nr de parametri */
        p->defptr = defptr;      /* Pointerul noii def */
        p->ref = 0;              /* Sterge semaforul ref */
        return(p);
    }
}

/*
 * Sterge un simbol din tabela de simboluri
 */

```

```

int sym_del(symbol)
char *symbol;
{
    register struct sym *p, *tp;
    register int bucket;

    bucket = bkt_map(*symbol);
    for ( p = sym_bkts[bucket];
        p != NIL;
        tp = p, p = p->next)
        if (lexeq(symbol, p->name)) {
            free(p->defptr); /* Elibereaza def veche */
            if (p == sym_bkts[bucket])
                sym_bkts[bucket] = p->next;
            else tp->next = p->next;
            free(p); /* .. si sterge nodul */
            return(OK);
        }
    return(ERROR);
}

/*
 * Suprapune un caracter alfabetic peste indexul
 * unei substructuri din tabela de simboluri
 */

int bkt_map(c)
    register char c;
{
    /* if(!c_alpha(c))
       screech("Invalid call to bkt_map()"); */
    if (c >= 'A' && c <= 'Z')
        return(c - 'A');
    if (c == '_')
        return(52);
    if (c == '$')
        return(53);
    return(c - ('a' - 26));
}

#ifdef PRT_SYMS

```

```

sym_print()    /* Listeaza continutul tabelii */
{
    register int i;
    register struct sym *p;
    register char *cp;

    printf("\n\nSymbol table dump follows:\n");
    for (i = 0; i < 54; i++)
        for (p=sym_bkts[i]; p != NIL; p = p->next) {
            printf("%s<%d,%d>=", p->name,
                    p->nargs, p->ref);
            for (cp = p->defptr; *cp != EOS; cp++)
                if (*cp & 0200)
                    printf("#%c", (*cp & 0177)+'0');
                else printf("%c", *cp);
            printf("\n");
        }
    return;
}
#endif

/* Init substructurile tabelii de simboluri */

sym_init()
{
    register int i;

    for (i = 0; i < 54; i++)
        sym_bkts[i] = NIL;
}

```

1.3. O bibliotecă de grafică în C

Utilizarea funcțiilor din biblioteca DAFC.OLB V1.1 (vezi și 1.3.8) asigură o interfață eficientă între programatorul C și cerințele dispozitivelor grafice din clasa DAF2020.

Majoritatea funcțiilor fac apel la alte funcții prezente în bibliotecă.

Aceste funcții ar putea fi clasificate după criterii ierarhice (utilizînd dimensiunea stivei necesare pentru CALL-RETURN), funcționale (ținînd seama de rolul jucat în transmisia către echipament sau în pregătirea trenului de comenzi ce face obiectul acestei transmisii) sau pragmatice

(folosind efectul apelului asupra programului și/sau echipamentului).

În cele ce urmează ele vor fi prezentate în lumina ultimului criteriu. Astfel, în bibliotecă există următoarele categorii de puncte de intrare:

- definire sesiune de desen;
- trasări/ștergeri;
- desenare texte;
- utilizare DAF2020 ca VT100;
- module speciale.

Prezentul manual nu descrie utilizarea unor puncte de intrare existente în bibliotecă dar care au rol intern.

Descrierea detaliată a celor de interes pentru programator (vezi și 1.3.8) face obiectul următoarelor secțiuni.

În sfârșit, să facem câteva precizări

hardware...

Suprafața de lucru a terminalului este următoarea:

- în mod Vt100, 24 de linii (a câte 12 pixeli, numerotate de sus în jos) cu câte 80 caractere pe linie (a câte 6 pixeli);
- în modul grafic, 512×384 pixeli din care 512×288 vizibili, suprapuși peste cadranul 1; din motive de compatibilitate a unor produse software create pentru TK4010, echipamentul consideră coordonatele unui pixel de afișat în limitele 0–1023 și 0–767 divizându-le intern la 2.

Lista switch-urilor ON la DAF–2020:1,6–9,14.

Lista switch-urilor ON la CDC–9335:(A)1, 6; (B)2, 3, 6; (C)5–7; (D)2, 3, 5, 10

Fără a suplini CARTEA TEHNICĂ a echipamentului (editată de FEPER), dăm mai jos lista unor comenzi speciale nefncapsulate în funcții:

- în modul de lucru TEKTRONIX:
 - 27, 12, 0 – șterge ecranul și pune cursorul în stînga, sus
 - 14, 0 – următoarele caractere editate vor fi mari
 - 15, 0 – următoarele caractere editate vor fi mici
 - 27, 88, 50, 0 – caracterele editate se suprapun
 - 27, 88, 51, 0 – caracterele editate șterg între matricea de 11×6 pixeli
- în modul de lucru VT100:
 - Scroll cu salt: 27,91,63,52,108,0
 - Scroll lin : 27,91,63,52,104,0
 - Ecran normal: 27,91,63,53,108,0
 - Ecran invers: 27,91,63,53,104,0

...și software

Biblioteca DAFC.OLB V1.1 face parte dintr-un grup de biblioteci care se adresează (separat) unor echipamente diverse (DAF2020, VDT52S, DGF1712) și/sau programatori în limbaje diferite (F77, C, PASCAL), dar care au comună (sau foarte asemănătoare) sintaxa majorității punctelor de intrare.

În diferite locuri este folosită noțiunea de variabilă de tip punct; este vorba despre o structură de date de 16 octeți tratată unitar pentru memorarea abscisei și ordonatei (ca numere reale reprezentate în dublă precizie) ale unui punct. Această structură este simulată în C prin intermediul altor structuri recunoscute de compilator (ex: tablouri de numere reale dublă precizie cu număr par de componente și cu abscisele pe locurile pare, etc.) sau utilizând facilitatea oferită de limbaj de a defini structuri de date proprii unei aplicații.

Manualul conține exemple de programe care apelează funcții din biblioteca DAFC.OLB.

O opțiune cu variate repercursiuni a fost — pentru toate versiunile — lipsa unui cod de retur (succes sau eroare) și a tratamentului specific valorii lui. Una din consecințe este că nici o funcție nu face verificarea corectitudinii parametrilor ce i se transmit, utilizatorul asumându-și responsabilitatea (și riscul !) pentru transmiterea incorectă a lor.

De asemenea utilizatorul este atenționat că următoarele date externe (inițializate și actualizate de funcțiile bibliotecii) nu trebuie distruse în timpul sesiunii de desen:

```
/*          DRWCOM          */

extern double X_min; /* coordonate fereastra */
extern double Y_min;
extern double X_max;
extern double Y_max;
extern double __sx; /* factori de scala */
extern double __sy;
extern double __cx; /* coordonate curente */
extern double __cy; /* (in mm) */
extern double __cr; /* param rotatiei */
extern double __sr;
extern double __xs; /* dimens text desenat */
extern double __ys;
extern double __ss;
extern double __ct; /* si inclinarea lui */
extern double __st;
extern int __lt; /* tipul curent de linie */
```

```
/*      DAFCOM      */
```

```
extern int  U_min; /* coordonate vizor */
extern int  V_min;
extern int  U_max;
extern int  V_max;
extern double __Q__;
```

Observația este valabilă și pentru variabilele globale următoare:

```
/*      FILCOM      */
```

```
extern double  _sa_ /* param drepteii de hasura */;
extern double  _ca_ /* in forma normala */;
extern double  _pp_ /* xcos(a)-ysin(a)=p */;
extern int     _mw_=64 /* dimens maxima */
                  /* a listei _ww_ */;
extern int     _iw_ /* indice curent in _ww_ */;
extern double  _ww_[64][2] /* puncte de inters */;
```

ultima fiind utilizată doar pentru hașurarea complexă (distrugerea ei produce rezultate necunoscute doar pentru hașurare).

În secțiunea 1.3.8 se prezintă un Memento pentru apelurile funcțiilor din bibliotecă DAFC.OLB.

Încă din acest punct atragem atenția că orice comparație între această bibliotecă și alte produse (eventual existente pe microcalculatoare) are o valoare strict sentimentală.

1.3.1. Definierea sesiunii de desen

Avînd în vedere neconcordanța dintre sistemul de axe al terminalului și cel al utilizatorului precum și disponibilitățile de interactivitate, DAFC.OLB conține următoarele puncte de intrare:

```
DAFINI
PIXEL
WCROSS, CROSS
SETWND, SETVP
SCALE, ROTAX
TYPLIN
CHANGE
DAFEND
```

DAFINI – Inițializare mod grafic

Apelul diverselor funcții din biblioteca DAFC.OLB are ca efect

lateral și actualizarea unor date globale dintre cele prezentate. Modulul **DAFINI** inițializează aceste date așa cum se precizează în descrierea punctelor de intrare care le actualizează.

Funcția se apelează sub forma

DAFINI(mode);

mode fiind un pointer la un șir de caractere conținând:

- "vt100" – pentru stabilirea modului inițial de lucru cu terminalul ca video terminal de tip Vt100 ...

- "Graphics" – ... sau de tip grafic

Schimbarea ulterioară a acestui mod se face cu funcția **CHANGE**.

Exemple de utilizare : EX01, EX02, etc.

Funcții apelate : **TYPLIN**, **CHANGE**.

PIXEL – aprindere pixel

Desenarea punctului de coordonate (ix, jy), ix=0,...,511 și jy=0,...,383, (cînd terminalul este utilizat ca TK, și ținînd seama de observația anterioară) se face cu apelul

PIXEL(ix, jy);

Exemple de utilizare : **CONJCT**

Funcții apelate : —

WCROSS – preluare coordonate în mm

Apelul **WCROSS(x, y, taste)** conduce la afișarea cursorului cruce și la inițializarea variabilelor reale x și y cu valorile abscisei și ordonatei locului de afișare a cursorului (în sistemul de axe al utilizatorului), precum și a variabilei caracter taste cu codul ASCII al tastei care a produs continuarea execuției.

Exemple de utilizare : —

Funcții apelate : **CROSS**

CROSS – preluare coordonate în pixeli

Apelul **CROSS(IX, JY, taste)** conduce la afișarea cursorului cruce și la inițializarea variabilelor întregi IX și JY cu valori (în pixeli) ale liniei (0,...,1023) și coloanei (0,...,767) pe care se află cursorul (după eventuale mișcări) și a variabilei caracter taste cu codul ASCII al tastei care a produs continuarea execuției.

SET WINDOW – definire fereastră

Limitarea reprezentării unor desene la porțiuni strict controlate ale universului este cerută în variate situații. Definirea acestei porțiuni ca

dreptunghi cu laturile paralele cu axele de coordonate ale universului se face cu apelul

SETWND(Xmin, Ymin, Xmax, Ymax);

parametrii fiind constante/variabile reale considerate în mm., care actualizează **X__min, Y__min, X__max, Y__max, __Q__**

După apelul **DAFINI** aceste variabile au valorile -90., -35., 90., 100., 5.67777.

Exemple de utilizare : CAR

Funcții apelate : —

SET ViewPort – definire vizor

Limitarea suprafeței de reprezentare este deasemenea utilă; definirea ei se face cu apelul

SETVP(Umin, Vmin, Umax, Vmax);

parametrii fiind constante/variabile întregi considerate în pixeli, care actualizează **U__min, V__min, U__max, V__max, __Q__**.

După apelul **DAFINI** aceste variabile au valorile 0, 0, 1023, 767, 5.67777.

Exemple de utilizare : CAR

Funcții apelate : —

ROTAX – Rotația axelor

Stabilirea unui unghi nenul (măsurat în sens trigonometric) între noile și vechile axe utilizator (rotația axelor), se obține cu apelul

ROTAX(ro);

ro – variabilă/constantă reală definind mărimea (în grade sexagesimale) a unghiului de rotație, și actualizînd **__ct** și **__st**. După **DAFINI** acest unghi este 0.

Exemple de utilizare : EX04.

Funcții apelate : —

SCALE – Scalarea

Unitatea utilizator de măsură a lungimii a fost stabilită, prin lipsă, milimetrul. Schimbarea ei (pentru ambele axe), se face cu apelul

SCALE(sx, sy);

sx – variabilă/constantă reală, reprezentînd lungimea în mm a noii unități de măsură a axei absciselor;

sy – analog, relativ la axa ordonatelor.

După apelul **DAFINI**, **__sx** și **__sy** sînt 1.

Observație: Fiind definite fereastra și vizorul, subrutinele de reprezentare realizează automat o scalare care să suprapună colțul stînga-jos al

ferestrei peste cel al vizorului, fără deformare.

Scalarea nu afectează coordonatele (în mm) ale ferestrei !

Exemple de utilizare : EX03, EX04, etc.

Funcții apelate : —

TYPLIN – setarea tipului de linie

Definirea modelului curent de linie se face inserînd în programul sursă linia

TYPLIN(m) ;

m fiind o variabilă/constantă întreagă reprezentînd tipul de linie utilizat:

- m < 0 linie întunecată
- m = 0 linie luminoasă continuă
- m = 1, 2 linie luminoasă întreruptă
- m = 3 linie luminoasă punctată
- m = 4 linie luminoasă linie-punct
- m > 4 este echivalent cu m = 0

După apelul DAFINI tipul curent de linie (___1t) este 0.

Exemple de utilizare : EX01.

Subprograme apelate : —

CHANGE – schimbarea modului de lucru

Schimbarea modului de lucru cu DAF 2020, în vederea utilizării de subrutine VT100 sau de grafică se face cu apelul

CHANGE(mode) ;

mode fiind pointer la un șir de caractere conținînd "vt100" sau "Graphics", cu semnificație evidentă.

DAFEND – sfîrșitul sesiunii de desen

Sfîrșitul sesiunii de desen se anunță cu apelul

DAFEND () ;

care produce și selecția modului VT100.

NOTĂ : *Modulele prezentei biblioteci se apelează în interiorul unei perechi de paranteze DAFINI/DAFEND; un task poate conține oricît astfel de perechi neconținute una în alta:*

Exemple de utilizare : EX01, EX02, etc.

Funcții apelate : —

1.3.2. Trasări/Ștergeri

În această secțiune se descrie:

MOVPEN

care „mișcă” efectiv spotul, realizându-se un desen, precum și

COPY

utilă pentru copierea ecranului pe imprimanta matricială CDC-9335.

Pentru ștergerea unui segment se apelează **TYPLIN** cu parametrul -1, apoi **MOVPEN** pentru trecerea peste segmentul de șters (în sensul desenării).

Pentru obținerea unor informații, se prezintă **WHERE** și **GETSTS**.

Dispozitivele **DAF** pot desena segmente de dreaptă (via un interpolator liniar existent hardware care unește punctele necesare de pe o rețea cu 512×384 pixeli).

MOVPEN

se apelează sub forma

MOVPEN (m, x, y);

unde

m este o mască (variabilă/constantă întreagă) de forma $10 \cdot P + C$, cu **P**, **C** luând valorile 0 sau 1:

- **P**(poziție toc) = 0 ține tocul sus (altfel, jos)
- **C**(coordonate) = 0 valorile indicate de **x**, **y** se consideră incrementări (altfel, coordonate) în sistemul de axe al dispozitivului. (Deci **m** poate lua valorile 0, 1, 10, 11: **m** > 2 va ține tocul sus, iar **m** < 2 va ține tocul jos; **m** par va interpreta **x**, **y** ca incrementări ale coordonatelor ultimului punct desenat, și nu coordonate ale punctului curent cum se întâmplă pentru **m** impar.)

x, **y** sînt variabile/constante reale reprezentînd coordonatele unui punct cu semnificație dependentă de **m**.

Coordonatele se consideră în sistemul utilizator, și se vor transmite către echipament alterate (modificate) de scalare, rotație și decupare la fereastra curent definită, dar se vor menține (în mm) în variabilele

cx, cy.

Exemple de utilizare : **EX01**, **EX02**, etc.

Funcții apelate : **CLIPP**, **HMOVE**.

WHERE – Coordonatele curente

Apelul

WHERE (x, y);

este adesea util într-un modul care nu a stabilit el însuși acest punct curent (și eventual trebuie refăcut înainte de **return** la chemător).

Notă: Alte informații utile pot fi receptate din datele globale al căror conținut a fost descris; este lăsat în responsabilitatea utilizatorului accesul (scriere neavizată !) la aceste date.

COPY – Copia ecranului

Apelul `COPY(dim)` produce copierea pe dispozitivul CDC-9335 cuplat ca hard-copy a conținutului memoriei-ecran, la dimensiunea indicată de constanta șir de caractere/taboul de elemente caracter `dim`, care ia valorile legale "Simple"/"Double".

GETSTS – Starea curentă a terminalului

Se află cu apelul

```
GETSTS(ix, jy, cod);
```

inițializându-se `ix` și `jy` cu coordonatele (în pixeli) ale cursorului, iar octetul `cod` conform definițiilor din cartea tehnică (vezi ESC ENQ).

1.3.3. Desenarea textelor

Editarea textelor este facilitată de următoarele funcții:

TEXTS

TEXTA

TEXT

Efectul apelului primelor două se menține până la un nou apel.

Desenarea textelor ține cont de sistemul de axe al utilizatorului (scalare, rotație) și de fereastra curent definită.

Funcția `VTXT` produce un text, exploatând facilitățile VT100 ale terminalului, și trebuie apelată în acest mod.

TEXTS – Dimensiunile textului

Apelul

```
TEXTS (ysize, xsize, ssize);
```

unde `ysize`, `xsize`, `ssize` sînt constante/variabile reale, declară înălțimea (____`ys`), lățimea (____`xs`) și spațiul dintre caracterele editate ulterior (____`ss`), ținînd seama de scalele (pentru `Ox` și `Oy`) curente. După apelul `DAFINI` aceste dimensiuni sînt respectiv 5., 5., 1.

Exemple de utilizare : `TXTTST`

Funcții apelate : —

TEXTA – Înclinarea textului

Înclinarea axei textului față de axa `Ox` a utilizatorului se face inserînd în program linia

```
TEXTA 'angle);
```

unde `angle` este variabila/constantă reală precizînd (în grade sexagesimale) înclinarea dorită și inițializînd ____`ct` și ____`st`.

După apelul `DAFINI` înclinarea inițială este 0.

Exemple de utilizare : TXTTST

Funcții apelate : —

TEXT – desenarea textului

Apelul

TEXT(text);

unde **text** este tablou de elemente **char**/constantă șir de caractere produce desenarea șirului "text" (terminat cu 0 (NUL)), și re poziționarea cursorului la începutul textului.

Exemple de utilizare : TXTTST.

Funcții apelate : **WHERE, MOVPEN.**

1.3.4. Modul VT-100

Următoarele funcții

ERASE

SCROLL

SETCP

SETCPS

LETCP

GETCP

SELECT

VTXT

TCOPY, NCOPY

facilitează exploatarea posibilităților oferite de DAF2020 de a lucra ca terminal de tip VT100. Ele se vor apela numai după ce în prealabil s-a apelat **CHANGE ("vt100")**.

Prin utilizarea succesivă a terminalului DAF2020 ca TEKTRONIX și VT100 se poate mări gradul de interactivitate al task-ului și se pot obține ecrane mixate care apoi să fie copiate pe imprimantă.

Trebuie totuși reținut că trecerea TK→VT se face cu anularea pixelilor de pe liniile mai mari decât 287.

Față de poziția curentă a cursorului se poate șterge o porțiune de ecran sau de linie. Aceasta se realizează cu apelul

ERASE(portiune, entitate);

Ambii parametri sînt pointeri la șiruri de caractere. **portiune** poate lua valorile:

- "All" pentru ștergerea întregii entități
- "Begin" pentru ștergerea entității de la început pîna în poziția curentă a cursorului
- "End" pentru ștergerea entității de la poziția curentă a cursorului pînă la sfîrșit

entitate poate lua valorile "Line" sau "Screen".

Exemple de utilizare : CLOCK

Funcții apelate : —

SCROLL – Defilarea

După umplerea ultimei linii terminalul face defilarea în sus pentru a face loc unei noi linii (pierzând prima linie de pe ecran).

Apelul

`SCROLL (linf, lsup);`

definește marginile de sus și de jos ale zonei pe care se produce defilarea. Ambii parametri sînt constante/variabile întregi (1,...,24).

Exemple de utilizare : CLOCK

Funcții apelate : —

SETCP – poziționare cursor

Facilitate specifică terminalelor cu tub catodic, poziționarea cursorului se realizează inserînd în program linia

`SETCP (ilin, jcol);`

unde `ilin` este constantă/variabilă întreagă cu valori de la 1 la 24; iar `jcol` este constantă/variabilă întreagă cu valori de la 1 la 80.

Pentru poziționarea cursorului în interiorul zonei de scroll se utilizează funcția `SETCP`; cei doi parametri definesc deasemenea linia și coloana destinație a cursorului.

Pentru poziționarea diferențială a cursorului se poate utiliza funcția `LETCP` cu următorii doi parametri:

- constantă de tip șir de caractere/variabilă—tablou de elemente char cu conținutul "Up" / "Down" / "Left" / "Right" și cu semnificație evidentă;
- constantă/variabilă întreagă și pozitivă indicînd numărul de linii (coloane) de deplasare a cursorului pe aceeași coloană (linie).

Exemple de utilizare : CLOCK

Funcții apelate: —

GETCP – citirea poziției curente a cursorului

Pentru a afla unde se află cursorul, se inserează în program linia

`GETCP (ilin, jcol);`

cu efectul inițializării variabilelor întregi `ilin / jcol` cu valorile liniei și coloanei pe care se află cursorul după o serie de mișcări.

Echipamentul poate afișa un text Normal (alb pe fond întunecat) sau Reversvideo (întunecat pe fond alb); în ambele cazuri textul poate fi

subliniat.

SELECT – selecția stilului de afișare

Selecția stilului de afișare se obține după apelul `SELECT(style)`, unde `style` este pointer la un text conținând "Normal", "Reverse" sau "Underlined".

Exemple de utilizare : `CLOCK`

Funcții apelate : —

VTXT – afișarea textului

După stabilirea poziției de început a textului și a modului de afișare, afișarea propriu-zisă se face pe calea

`VTXT (dim, txt);`

unde cei doi parametri sînt pointeri la șiruri de caractere; primul poate lua valorile "Normal", "Double" sau "Large" pentru a stabili mărimea caracterelor textului, iar al doilea conține textul de afișat.

Se consideră terminator de text caracterul 0(NUL), pus implicit de compilator pentru constante.

Exemple de utilizare : `CLOCK`

Funcții apelate : `GETCP`, `SETCP`.

HARD-COPY : copierea ecranului la imprimantă

Pentru a permite (sau a interzice) transmisia caracterelor venite de la calculator spre ecran și către imprimantă CDC-9335, se inserează în program linia

`TCOPY();`

sau

`NCOPY();`

Deoarece codurile de control ale celor două dispozitive sînt diferite, se obțin rezultate nedorite dacă se lucrează în mod transparent (`TCOPY`) și se transmit și către imprimantă coduri de control specifice ecranului.

1.3.5. Funcții specializate

Aceste funcții sînt de interes specializat. Setul acestor funcții poate fi îmbogățit de fiecare utilizator, după necesitățile locale. În prezent, biblioteca DAFC.OLB conține

`INSEG, INPLG`
`CLIPP, USRWND, UCLIPP`
`LINE`
`GRAPH`

CERCRC, POLREG
ARCC3P, CERC3P
FILL, LDSEG, LDPLG

Test de apartenență

Date fiind un punct P și un poligon cu n vîrfuri memorate în lista w de n puncte, funcția

`INPLG(P, n, w)`

`double P[2], w[][2]; int n;`

returnează valorile:

- -1, dacă P este exterior poligonului w
- 0, dacă P se află pe frontiera poligonului
- +1, dacă P este interior;

Dacă u este o listă de 3 puncte cu semnificația punct inițial, punct intermediar și final, funcția `INSEG(P, u)` returnează valorile -1, 0, +1 în conformitate cu poziția punctului P față de segmentul de cerc precizat de u .

Decupare

Presupunînd definită o fereastră ca poligon (convex) cu maximum 6 colțuri, apelul

`UCLIPP(P, Q);`

`double P[2][2], Q[2][2];`

produce un cod de retur și coordonatele subsegmentului vizibil Q al segmentului P (liste de două variabile de tip punct). Codul de retur are semnificațiile următoare:

- 0 segmentul P complet vizibil
- 4 segmentul P invizibil
- 1 vizibil subsegmentul $Q[1]P[2]$
- 2 vizibil subsegmentul $P[1]Q[2]$
- 3 vizibil subsegmentul $Q[1]Q[2]$

Secvența de apel pentru `USRWND` este

`USRWND(n, wnd);`

primul parametru fiind constantă/variabilă întreagă indicînd prin valoarea ei absolută numărul de vîrfuri ale ferestrei; dacă este negativă se omite desenarea ferestrei. Al doilea parametru este variabilă de tip listă de puncte ale poligonului wnd .

Decuparea față de fereastra standard (definită cu `SETWND`) se face cu apelul `CLIPP (P, Q)`; parametrii au aceeași semnificație ca mai sus.

Grafice de funcții continue

Dacă a, b sînt numere reale, $a < b$ și $f: [a, b] \rightarrow \mathbb{R}$ este o funcție continuă al cărei grafic se cere desenat, acest lucru se obține pe calea

`GRAPH(f, a, b);`

Note:

1) Este legal și un apel de forma `GRAPH(f, b, a)` cu efectul corespunzător.

2) Funcția f trebuie să fie subiectul unei declarații "`double f()`" în unitatea de program apelantă a funcției `GRAPH` (vezi și exemplele).

Exemple de utilizare : EX03, EX04

Funcții apelate: `MOVPEN`

Desenarea cercurilor cu raza și centrul date

`CERCRC(r, xc, yc);`

produce desenarea unui cerc de rază r și centru (xc, yc) . Toți parametrii sînt reali.

Desenarea unui poligon regulat cu n laturi înscris în cercul cu centrul în (xc, yc) și avînd un vîrf în punctul curent se face prin apelul

`POLREG(xc, yc, n);`

Exemple de utilizare : EX02

Funcții apelate : `WHERE, MOVPEN`

Dacă se dorește desenarea unui arc de cerc cunoscînd coordonatele punctelor extreme și cele ale unui punct intermediar, se poate folosi apelul

`ARCC3P(A, B, C);`

A, B, C sînt punctele inițial, intermediar și final definoitoare pentru arcul de cerc dorit. Punctul intermediar se consideră apartenent arcului.

Funcții apelate: `MOVPEN`

Hașurarea complexă

Fie date $np1$ poligoane de dimensiuni $lp1[0], lp1[1], \dots, lp1[np1-1]$ și cu vîrfurile memorate în continuare în lista $p1$ care se doresc hașurate; fie deasemenea $np0$ poligoane de dimensiuni $lp0[0], lp0[1], \dots, lp0[np0-1]$ cu vîrfurile memorate în lista $p0$ care nu se vor hașura;

Fie $nc1$ segmente de cerc definite de cîte trei puncte memorate consecutiv în lista $c1$ care se doresc hașurate și $nc0$ segmente de cerc memorate în $c0$ care să nu se hașureze.

În această organizare, hașurarea structurii definite de coordonatele date în tablourile $p1, p0, c1, c0$ cu drepte paralele care fac unghiul

"chi" (măsurat în sens trigonometric, în grade) cu axa Ox și echidistanța dintre ele "step", se realizează cu apelul

FILL

```
(np1, /* numarul poligoanelor incluse in hasura */
lp1, /* lista dimensiunilor acelor poligoane */
p1, /* lista coord tuturor polig incl in has*/
np0, /* numarul poligoanelor excluse din hasura*/
lp0, /* ... si lista dimens lor */
p0, /* lista coord tuturor polig excl din has*/
nc1, /* numarul sectoarelor de cerc din hasura*/
c1, /* si lista celor nc1*3 puncte */
nc0, /* numarul sect excluse din hasurare */
c0, /* si lista celor nc0*3 puncte definitorii*/
step, /* pasul dintre doua linii de hasura*/
chi /* unghiul (masurat in grade, in sens trig)
dintre axa absc si dreapta de hasura */);
```

. Dacă se obțin mai mult de 64 puncte de intersecție, cele în exces se ignoră (ceea ce se vede pe figură); evitarea acestei situații se face definind explicit în modulul chemător `__mw__` mărit (el este stabilit la 64), și `__ww__` alocat adecvat.

Încărcarea listei `__ww__` cu punctele de intersecție ale dreptei de hașură cu un poligon oarecare sau cu un segment de cerc dat se face folosind funcțiile LDPLG și LDSEG. Dacă se definesc corect datele din FILCOM, ele pot fi folosite independent, astfel:

```
double p[][2], s[][3][2];
LDPLG(n, p);
LDSEG(s);
```

unde p este identificatorul poligonului cu n laturi, iar s este o listă de 3 puncte ce definesc un segment de cerc.

1.3.6. Exemple

Scopul acestei secțiuni este să prezinte în cazuri concrete exemple de apel al funcțiilor din biblioteca DAFC.OLB; programele respective nu au nevoie de mai multe explicații decât cele din comentariile încorporate. Deși utilizează un minim de cunoștințe și/sau modele matematice, scopul principal rămâne exemplificarea modului de folosire a funcțiilor din biblioteca DAFC.OLB pentru obținerea unui desen. Pentru comparații utile vezi [5].

Programul CONJCT

```

/*
 *
 * Programul CONJCT rezolva problema lasata
 * deschisa in articolul "Asupra unei conjecturi"
 * din GM-8/1984
 *
 * Autor: Adrian Posea
 */
double a=288.;
int    linfx=0,lsupx=143,linfy=0,lsupy=143;
double eps1=1.e-5,eps2=1.e-6,eps3=1.e-7;
$$narg=1;
main()
{
  int    i,j;
  double r,q,ri,rj,p,a1,a2,a3,a4;
  double x1,y1,x2,y2,x3,y3,x4,y4;
  double sqrt(),abs();
  double o1o2,o1o3,o1o4,o2o3,o2o4,o3o4;
  DAFINI("V"); ERASE("All", "Screen");
  XMIT("\0331\035");
  for(i=linfx;i<lsupx;i++){
    r=i;r=r*r;
    ri=(a-i)*(a-i);
    for(j=linfy;j<lsupy;j++){
      q=j;q=q*q;
      rj=(a-j)*(a-j);
      a1=sqrt(r+q);
      a2=sqrt(r+rj);
      a3=sqrt(ri+rj);
      a4=sqrt(ri+q);
      p=1.0+(a1+a2)/a;
      x1=i/p; y1=(j+a1)/p;
      p=1.+(a2+a3)/a;
      x2=(i+a2)/p;y2=(j+a2+a3)/p;
      p=1.+(a3+a4)/a;
      x3=(i+a3+a4)/p;y3=(j+a4)/p;
      p=1.+(a4+a1)/a;
      x4=(i+a1)/p;y4=j/p;
    }
  }
}

```

```

o1o2=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
o1o3=sqrt((x3-x1)*(x3-x1)+(y3-y1)*(y3-y1));
o1o4=sqrt((x4-x1)*(x4-x1)+(y4-y1)*(y4-y1));
o2o3=sqrt((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2));
o2o4=sqrt((x4-x2)*(x4-x2)+(y4-y2)*(y4-y2));
o3o4=sqrt((x4-x3)*(x4-x3)+(y4-y3)*(y4-y3));
p=(o1o2*o3o4+o2o3*o1o4)/(o1o3*o2o4);p=abs(p-1.);
if(p < eps1) PIXEL(i,j);
if(p < eps2) PIXEL(i+160,j);
if(p < eps3) PIXEL(i+320,j);
}}
COPY("Simple");
DAFEND();
}

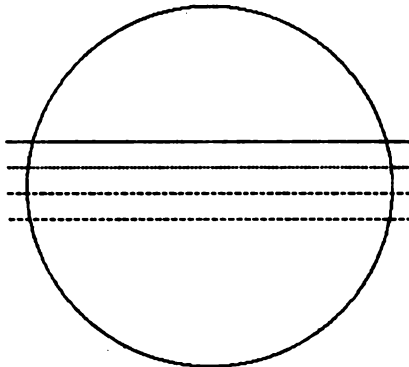
```

Modelele de linie

```

/*
C
C      EX01
C      Desenarea, pentru referinta,
C      a celor 4 modele de linie
C
C      */

```



```

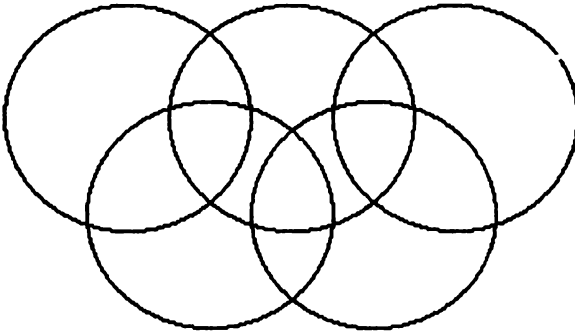
main()
{ int i; double s;
  s = 100.;
  DAFINI("Graphics");
  CERCRC(40.,5.,10.);
  movpen(1,-45.,45.);

```

```
for(i=1; i<8; i++) {  
  TYPLIN(i-1);  
  MOVPEN(10, s, 0.);  
  MOVPEN(00, 0., -10.);  
  s = -s; };  
COPY("S");  
DAFEND();  
}
```

Cercurile Olimpice

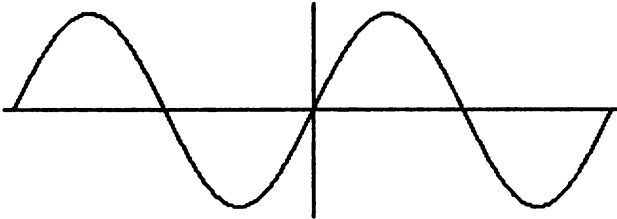
```
main()  
{  
  /*  
  C      EX02  
  C      Cinci cercuri olimpice  
  C  
  */
```



```
DAFINI("Graphics");  
CERCRC(20., -30., 25.);  
CERCRC(20., 0., 25.);  
CERCRC(20., 30., 25.);  
CERCRC(20., -15., 0.);  
CERCRC(20., 15., 0.);  
DAFEND();  
}
```

Graficul funcției SINUS

```
/*
C      EX03
C Graficul functiei sin:[A, B]->[-1., 1.]
C
*/
```



```
main()
{  double sin(), a,b;
    b  = 2. * 3.141592;
    a  = -b;
    DAFINI("Graphics");
    SCALE(10., 10.);
    movpen(1,a,0.); movpen(11,b,0.);
    movpen(1,0.,1.);movpen(11,0.,-1.);
    GRAPH(sin, a, b);
    DAFEND();
}
```

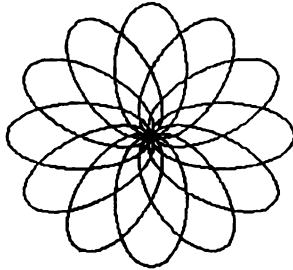
O floare

```
main()
/* PROGRAM EX04
C
C Floare obtinuta prin rotirea unei elipse
C
*/
{ int i; double f(), g();
  dafini("Graphics");
  scale(2., 2.);
  for(i=0; i<12; i++) {
    graph(f,0.,16.);
```

```

    movpen(1,0.,0.);
    graph(g,0.,16.);
    movpen(1,0.,0.);
    rotax(30.);};
    dafend();
}

```



```

double f(x)
double x;
/*
C
C Ecuatia unei jumatati de elipsa...
C
*/
{ double sqrt(), abs();
  return (4.*sqrt(abs(1.D0-((x-8.)*(x-8.)/64.))));
}
double g(x)
double x;
/*
C
C...si a celeilalte jumatati
C
*/
{return (-f(x));}

```

Programul TXTTST

```

char *c[]= {" ordonata spira,",
            " Sunet",
            " Fruct de lira,",
            " Capat paralogic",
            " Leagan mitologic"};

```

```
main()
/*
    Program TXTTST
C
C Utilizarea subrutinelor de editare texte
C
*/
```

Ordonata spiră,

Fruct de liliac,

Sunet

u r e o t e l e d e p e r e u

u r e o t e l e d e p e r e u

Suflet și grăi,

Vatra de traie,

d a r u l c e l e d a i

u r e o t e l e d e p e r e u

u r e o t e l e d e p e r e u

```
{ register int i;
  DAFINI("Graphics");
  MOVPEN(1,15.,10.);
  for(i=0; i; i++){
    TEXTA(45.*i);
    TEXTS(3.,4.,5.-i);
```



```

    TEXT(c[i]);};
    DAFEND();
}

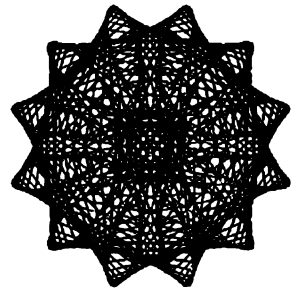
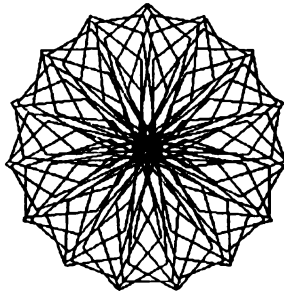
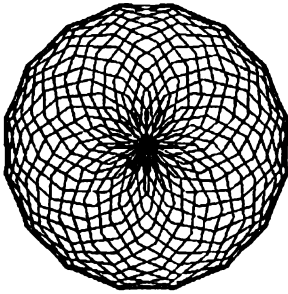
```

Program TRANDAFIR

```

/*
    "A rose is a rose"  TMM/7/1987 (revedeti [5])
*/

```



```

$$narg=1;
main()
{
    long n, d;
    scanf("%d%d",&n, &d);
    dafini("G"); scale(20., 20.);
    rose(n, d);
    dafend();
}
rose(n, d)
long n, d;
{
    long i, j, k;
    double s(), c(), r;
    k=0; i=0;
    while(k<360) {
        r=s(n*i); movpen(1, r*c(i), r*s(i));
        j=i+d;
        while(j!=i){
            j %= 360;
            r = s((n*j) % 360);
            movpen(11, r*c(j), r*s(j));

```

```

        k++; j += d;}
    i++;}
}
double s(i)
long i;
{
    double sin();
    return sin(i*3.141592/180.);
}
double c(i)
long i;
{
    double cos();
    return cos(i*3.141592/180.);
}

```

Programul SIERPINSKI

```

/*
    Program SIERPINSKI
    prezentat pentru T-grafica si apelul recursiv
*/
$$narg=1; /* inhiba preluarea unei linii de comanda */
main()
{
    double sx, sy;
    scanf("%lf,%lf",&sx,&sy);
    trtl_start();
    setwnd(0.2,0.2,102.3*2.,76.7*2.);
    scale(sx, sy);
    SIERP(3.,4);
    trtl_stop();
}
SIERP(s,level)
double s;
int level;
{
    int i;
    movpen(1,0.,0.);
    for(i=0;i<4;i++)
        { /* desenarea unui sfert de curba */
            side(level,s);
        }
}

```

```

        right(45.);
        forwrd(s/1.4142);
        right(45.);
    }
}
side(level,s)
int level;
double s;
{
    int l;
    double d;
    d=s/1.4142;
    if(level!=0) {
        l=level-1;
        side(l,s); right(45.);
    }
    forwrd(d); right(45.);
    side(l,s); left(90.); forwrd(s); left(90.);
    side(l,s); right(45.); forwrd(d); right(45.);
    side(l,s);
}
}
/* Urmatoarele functii realizeaza - in mic -
   filozofia TURTLE, proprie limbajului LOGO
   Vom prezenta chiar si functii care nu au fost
   apelate pentru eventuale comparatii cu
   procedurile scrise in limbajul PASCAL.
*/
static double dir =90.;
static double dirs=1.0;
static double dirc=0.0;

trtl_start()
{
    dafini("G"); movpen(1, 0., 0.);
}

right(u)
double u;
{
    double sin(),cos();
    dir-=u;
    u=dir*(3.141592/180.0);
    dirs=sin(u); dirc=cos(u);
}

```



```

    }

    left(u)
    double u;
    {
        right(-u);
    }

    forwrd(a)
    double a;
    {
        double x,y;
        x=a*dirc;
        y=a*dirs;
        movpen(10,x,y);
    }

    backwd(a)
    double a;
    {
        forwrd(-a);
    }

    setX(x)
    double x;
    {
        movpen(10, x, 0.);
    }

    setY(y)
    double y;
    {
        movpen(10, 0., y);
    }

    setpos(x, y)
    double x, y;
    {
        movpen(10, x, y);
    }

    setrep(x0, y0)
    double x0,y0;

```

```

{
    movpen(1, x0, y0);
}

trtl_stop()
{
    dafend();
}

```

Programul MAȘINA

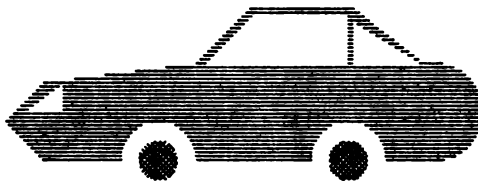
```

/*
    CAR
    Desenarea unei masini
    de curse... informatice

    Autor: A. Posea
*/
$$narg=1;
int    lp1[]={8},lp0[]={4,3,3};
double pl[]={0.,3.,3.,6.,8.,6.,12.,3.,
              12.,-2.,-8.,-2.,-10.,0.,-8.,2.};
double p0[]={.5,3.,3.,5.5,8.,5.5,8.,3.,
              8.2,3.,11.8,3.,8.2,5.2,
              -9.,.5,-7.,2.,-7.,.5};
double al[][3][2]={12.,3.,15.,0.,12.,-2.};
double cl[][3][2]={7.,-2.,8.,-1.,9.,-2.,
                  9.,-2.,8.,-3.,7.,-2.,
                  -1.,-2.,-2.,-1.,-3.,-2.,
                  -3.,-2.,-2.,-3.,-1.,-2.};
double a0[][3][2]={0.,-2.,-2.,0.,-4.,-2.,
                  6.,-2.,8.,0.,10.,-2.};

```

My car



```
main()
{
    dafini("Vt100"); erase("All", "Screen");
    setcp(5,25); vtxt("Large","My Car");
    change("Graphics");
    setvp(300,50,770,285); setwnd(-20.,-10.,20.,10.);
    FILL(1,lp1,p1,3,lp0,p0,1,a1,2,a0,.2,0.);
    FILL(0,0,0,0,0,0,4,c1,0,0,.2,45.);
    FILL(0,0,0,0,0,0,4,c1,0,0,.2,135.);
    dafend();
}
```

Programul CEAS

```
/*
c  CLOCK
c  (exemplu simplu de animatie)
c
c  Autor: A. Posea
*/
$$narg=1;
main()
{
    int    i, j, l[3], time[6];
    double sin(), cos();
    double p1[2], p2[2], x[3], y[3];
    double u, su, cu, oldu[3], newu[3];
    double v, sv, cv, r1, r2;
    l[0]=17; l[1]=23; l[2]=28;
    r1=0.104719; r2=0.523599;
    oldu[0]=oldu[1]=oldu[2]=-1.;
    DAFINI("Vt100"); ERASE("All", "Screen");
    SETCP(24, 15);
    SELECT("Revers");
    VTXT("Double", "Clock made by AP");
    SELECT("Normal");
    CHANGE("Gra");
    SETWND(-90.,-55.,90.,80.);
    for (su=43.; su<45; su+=.1){
        MOVPEN(1, 0., su);
        POLREG(0., 0., 8);
    }
}
```

```

for(i=0, u=0.; i<12; i++, u+=r2) {
    su=sin(u); cu=cos(u);
    p1[0]=34.*su;p1[1]=34.*cu;
    p2[0]=30.*su;p2[1]=30.*cu;
    LINE(p1, p2);
    for(j=0, v=u+r1; j<4; j++, v+=r1) {
        sv=sin(v); cv=cos(v);
        p1[0]=34.*sv;p1[1]=34.*cv;
        p2[0]=33.*sv;p2[1]=33.*cv;
        LINE(p1, p2);};
    }
p1[0]=0.;p1[1]=0.;
for(;;){ GTIM(time);
if(time[3] > 12) time[3]-=12;
newu[0]=(time[3]*5+time[4]/12)*r1;
newu[1]=time[4]*r1;
newu[2]=time[5]*r1;
if(oldu[0] != newu[0]) i=3; else
    if(oldu[1] != newu[1]) i=2; else
        if(oldu[2] != newu[2]) i=1; else
i=0;
if(i>0) {TYPLIN(-1);
    for (j=0; j<i; j++) {p2[0]=x[2-j]; p2[1]=y[2-j];
        LINE(p1, p2);};
    TYPLIN(0);
};
    for (j=0; j<3; j++) {oldu[j]=newu[j];
        p2[0]=(x[j]=(double)l[j]*sin(newu[j]));
        p2[1]=(y[j]=(double)l[j]*cos(newu[j]));
        LINE(p1, p2);};
    }
}
LINE(p1,p2)
double p1[2], p2[2];
{
    MOVPEN(01, p1[0], p1[1]);
    MOVPEN(11, p2[0], p2[1]);
}

```

1.3.7. Probleme propuse

- 1) Scrieți o funcție (recursivă) care să producă în ordinea citirii (de la stînga la dreapta) cifrele unui număr întreg dat.

- 2) Scrieți un program care poate tipări (după orice cuvânt preluat dintr-un fișier de intrare) lista cuvintelor distincte în ordinea descrescătoare a frecvenței de apariție.
- 3) Dacă fiind o zonă de memorie `_HEAP_`, cu `HEAP_S` unități libere, să se scrie funcțiile care
 - returnează adresa descrierii primului bloc de minim `n` unități libere ce urmează a fi utilizate;
 - declară liber blocul descris la adresa `bp` dată, și efectuează compactarea blocurilor libere vecine.

1.3.8. Observații complementare

Obținerea unui Task

Biblioteca `DAFC.OLB` este utilă programatorilor în C.

Linia de comandă pentru TKB va conține această bibliotecă înaintea celei cu modulele `OTS` invocate de compilatorul C, pentru a satisface algoritmul de tratare a simbolurilor referite și definite.

Prin urmare, obținerea de task-uri din exemplele de mai sus se face pe calea:

```
TKB>
TKB>EX01/FP/CP=EX01, LB:[ 1, 1]DAFC/LB, C/LB
TKB>/
Enter options:
TKB>STACK=2000 ; sau adecvat
TKB>...
TKB>//
```

Utilizarea funcției XMIT

Toate modulele prezentei biblioteci fac apel (direct sau indirect) la serviciile subrutinei `XMIT`. O secvență corectă de apel este

```
char *buf;
.
XMIT (buf);
.
```

și produce transmiterea către terminalul asociat cu numărul logic `.MOLUN` a tuturor caracterelor din `buf` până la primul egal cu `0(NUL)`.

Biblioteca `C.OLB` conține funcția `msg` care apelată cu parametrul `buf` îl transmite pe `stderr`. Am preferat pe `XMIT` din motive de compatibilitate cu `DAFF77.OLB`, și pentru independența de LUN-urile `1...n`, unde `n` este stabilit de opțiunea (cventual implicită) `TKB>UNITS=n`.

Memento DAFC

```

/* inceput sesiune desen */
    dafini(mode)
        char    *mode;

/* sfirsit sesiune desen */
    dafend()

/* arc de cerc dat prin 3 puncte */
    arcc3p(u, v, w)
        double  u[], v[], w[];

/* decuparea segmentului p; se obtine segmentul q */
    clipp(p, q)
        double  p[][2], q[][2];

/* stabilire fereastră */
    setwnd(xmin,ymin,xmax,ymax)
        double  xmin,ymin,xmax,ymax;

/* hasurare complexa */
    fill(np1, lp1, p1, np0, lp0,
        p0, nc1, c1, nc0,c0,
        step, chi)
        int      np1, lp1[]; double p1[][2];
        int      np0, lp0[]; double p0[][2];
        int      nc1, nc0;
        double  c1[][3][2], c0[][3][2];
        double  step, chi;

/* trasarea graficului functiei f:[a, b]-R */
    graph(f, a, b)
        double  f(), a,b;

/* determinare pozitie p fata de poligonul w */
    inplg(p, n, w)
        double  p[2], w[][2];
        int      n;

/* determinare pozitie p fata de segm de cerc u */
    inseg(p, u)
        double  p[2], u[][2];

```

```

/* stabilire vizor */
    setvp(umin,vmin,umax,vmax)
        int      umin,vmin,umax,vmax;

/* miscare penel */
    movpen(k, xfin, yfin)
        int      k;
        double   xfin, yfin;

/* unde este penelul ? */
    where(xw, yw)
        double   *xw, *yw;

/* desenare poligon regulat */
    polreg(xc, yc, npct)
        double   xc, yc;
        int      npct;

/* cerc dat prin raza si centru */
    cercrc(r, xc, yc)
        double   r, xc, yc;

/* rotirea axelor */
    rotax(alpha)
        double   alpha;

/* scalarea */
    scale(xscale, yscale)
        double   xscale, yscale;

/* desenarea unui text */
    text(t)
        char     t[];

/* stabilire unghi pt axa text */
    texta(a)
        double   a;

/* stabilire marime text */
    texts(ys, xs, ss)
        double   ys, xs, ss;

```

Sînt incluse și următoarele funcții cu rol de calcul matematic:

```
/* */ sign(x)
double x;

/* */ double abs(x)
double x;

/* calcul determinant */ double det(x, a, b)
double x[2], a[2], b[2];

/* */ double min(x, y)
double x, y;

/* */ double max(x, y)
double x, y;

/* */ double sin(x)
double x;

/* */ double cos(x)
double x;

/* */ double exp(x)
double x;

/* */ double ln(x)
double x;

/* */ double atan2(y, x)
double y, x;
```

Următoarele funcții sînt apelabile pentru exploatarea facilităților terminalului DAF-2020:

```
/* schimbare mod de lucru */
change(mode)
char *mode;

/* copiere ecran pe CDC-9335 */
copy(arg)
char *arg;
```

```

/* face CDC-9335 hard-copy */
hcopy()

/* interzice hard-copiarea */
ncopy()

/* tipul de linie curenta */
typlin(t)
    int t;

/* citire pozitie cursor grafic */
cross(i,j,t)
    int *i, *j;
    char *t;

/* desenare pixel */
pixel(i,j)
    int i,j;

/* sterge ecran */
erase(fct,arg)
    char *fct, *arg;

/* stabilire poz cursor alfa */
setcp(i,j)
    int i,j;

/* */
letcp(arg,inc)
    char *arg;
    int inc;

/* stabilire limite zona defilare */
scroll(linf,lsup)
    int linf,lsup;

/* */
setcps(i,j)
    int i,j;

/* citire pozitie cursor alfa */
getcp(i,j)
    int *i,*j;

```

```

/* selectie mod afisare alfa */
    select(style)
        char    *style;

/* afisare text VT100 */
    vtxt(style,txt)
        char    *style,txt[];

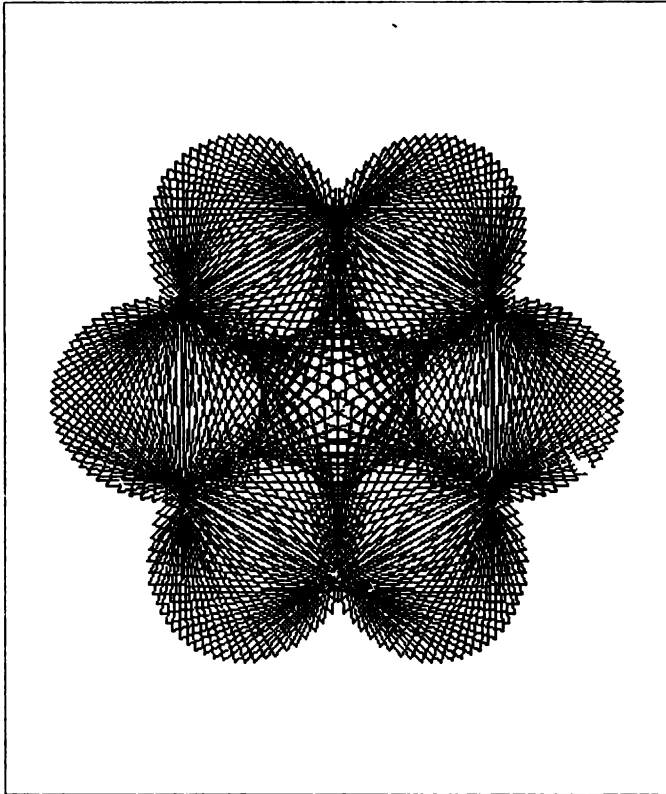
/* */ xmit(string)
    char    string[];

```

1.4. Bibliografie

- [1] Theodosias PAVLIDIS
Algorithms for graphics and image processing, Springer Verlag, 1982
- [2] Dan ROMAN, Adrian LUSTIG, Constantin STĂNESCU
Algoritmi de automatizare a proiectării, Ed Militară, 1988
- [3] James FOLEY, Andries Van DAM
Fundamentals of interactive computer graphics, Addison-Wesley Publ Co, 1981
- [4] A T BERZTIS
Data Structures – Theory and Practice, Academic Press, 1971
- [5] Marin VLADA, Adrian POSEA
Grafică automată în limbajul FORTRAN77 și aplicații, Tipografia Universității București, ediția II, 1990
- [6] Brian KERNIGHAN, Denis RITCHIE
The C programming language, Prentice-Hall, 1978
- [7] Martin MINOW
DECUS C LANGUAGE SYSTEM – Compiler & Library Software Support Manual, SIG 1981

GRAFICĂ ÎN LIMBAJUL PASCAL (OREGON)



| | |
|---|-----------|
| 2. GRAFICĂ ÎN LIMBAJUL PASCAL (OREGON) | 61 |
| 2.1. Utilizarea limbajului PASCAL (memento) | 65 |
| 2.2. Exemple de programe PASCAL | 65 |
| 2.3. O bibliotecă de grafică în PASCAL | 69 |
| 2.3.1. Definirea sesiunii de desen | 71 |
| 2.3.2. Trasări/Ștergeri | 74 |
| 2.3.3. Desenarea textelor | 75 |
| 2.3.4. Modul VT-100 | 76 |
| 2.3.5. Subrutine specializate | 78 |
| 2.3.6. Exemple | 80 |
| 2.3.7. Probleme propuse | 84 |
| 2.3.8. Observații complementare | 84 |
| 2.4. Bibliografie | 85 |

Limbajul PASCAL a fost conceput pentru învățarea informaticii: structuri de date, de control și stil de programare; N. Wirth publica în 1971 în „Acta Informatica” [1] articolul „The programming language PASCAL”, iar în 1974 — la Prentice Hall — cartea „Algorithms + Data Structures = Programs”. În martie 1978 ISO a votat standardizarea lui.

Iată o listă a câtorva caracteristici ale limbajului PASCAL:

- instrucțiunile se scriu pe mediul de intrare fără nici o restricție; se utilizează ‘;’ ca separator (ca în C)
- unor cuvinte li se recunoaște o semnificație specială (repeāt, var, etc) și nu pot fi folosite ca identificatori (ca în toate limbajele de programare)
- declararea tuturor obiectelor cu care se lucrează (date și proceduri) este obligatorie (ca în COBOL); nu se admit declarații implicite
- tipurile standard de date recunoscute sînt: octet, număr întreg, număr real; facilitățile de structurare proprii limbajului permit construirea de structuri de date oricît de complexe: tablouri (care trebuie alocate static; nu există ceva de tipul **VIRTUAL** din FORTRAN77), înregistrări (ca în COBOL sau PL/I), mulțimi (ca bit string în PL/I) și fișiere (secvențiale).
- pentru manipularca unor date alocate dinamic există procedurile standard **new** și **dispose** care fac apel la adresele și nu la numele variabilelor utilizate
- instrucțiunile de bază sînt atribuirea, execuția condiționată (de diverse forme), etc; există instrucțiuni pentru construirea de structuri de control complexe: gruparea în blocuri (**begin-end** în ALGOL, **do** în PL/I, etc)
- procedurile și funcțiile se pot chema recursiv: parametrii se transmit numai prin valoare sau adresă (niciodată prin nume)
- operațiile de intrare-ieșire sînt foarte slab elaborate: nici absente (ca în C, urmînd ca problema să fie rezolvată de implementatorul compilatorului), nici prezente (ca în COBOL sau FORTRAN 77 unde există o varietate de posibilități de lucru cu fișiere).

În PASCAL se impun reguli stricte de concordanță între operatori și tipul operanzilor asupra cărora acționează, în scopul depistării încă din faza de compilare a erorilor, ca reacție la diverse tipuri de erori obținute în programele compilate cu FORTRAN.

În PASCAL este posibilă utilizarea variabilelor de tip referință dar se folosesc mai ales în legătură cu date alocate dinamic și care nu pot fi referite prin nume simbolice, spre deosebire de C unde este dezvoltată o adevărată aritmetică a pointerilor.

Structura monolitică a programului compilat cu PASCAL (adică făcut dintr-un program principal ce include un număr de proceduri care pot și ele să includă alte proceduri, etc.) prezintă avantajul unor puternice verificări, dar îngreunează lucrul în echipă (mult facilitat de C) și chiar lucrul cu biblioteci de module obiect.

Limbajul PASCAL nu recunoaște noțiunea de fișier extern, deși practic nu există program nebanal care să nu folosească fișiere de date.

Limbajul PASCAL nu are definită noțiunea de zonă de date comune (ca în FORTRAN, sau macăr ca în C). Această lipsă este desigur suplinită de programatorii avansați pe căi ocolite, artificial.

Multe alte diferențe față de limbajele de programare clasice au impus implementatorilor mediului PASCAL să recurgă la extensii față de standard, care tind să apropie limbajul PASCAL de limbajul C: posibilitatea de a folosi proceduri și funcții standard pentru diferite operații (lucrul cu fișiere, cu șiruri de biți, `sizeof`), compilarea separată a unor bucăți din programul sursă (cu obținerea eventual a unui fișier .MAC în locul unuia .OBJ pentru facilitarea inserției de cod în limbaj de asamblare), etc.

Tonul relativ critic cu care a fost prezentat limbajul PASCAL nu este generat de caracteristicile lui vis-a-vis de pretențiile inițiale (limbaj de învățare a informaticii), ci de faptul că el a devenit limbaj folosit la nivel comercial și continuă să crească prin a-și trăda originea (ca și BASIC). Aceste fapte nu trebuie să ne facă să nu observăm că mediul PASCAL a fost folosit pentru dezvoltarea de programe foarte mari, iar compilatorul s-a dovedit compact și eficient și în aceste cazuri, nu doar pentru programele-școală mici, care utilizează doar câteva construcții. În plus, chiar dacă programele PASCAL nu vor fi total portabile între diferite medii PASCAL, compilatorul este suficient de portabil între diferite configurații hardware, ceea ce va induce o bună portabilitate și pentru programele utilizatorilor.

Nu știm cât de realistă este acțiunea „esperanto”, dar o (re)acțiune de tip PL/I este necesară: e nevoie de un limbaj pentru calcule economice, de altul pentru calcule tehnico-științifice, pentru învățarea informaticii și desigur de unul pentru scrierea mediilor limbajelor amintite. Dincolo de nuanțe, și având în vedere experiența de până acum a celor care lucrează

simultan în COBOL, FORTRAN, PASCAL, C se poate afirma că un singur limbaj (PL/II ?) ar fi suficient.

2.1. Utilizarea limbajului PASCAL (memento)

Analizorul lexical al compilatorului PASCAL extrage din textul-sursă tokenii proprii (simboluri speciale, identificatori, numere, etichete și șiruri de caractere) ignorând spațiile albe sau prezentând mesaje de eroare în rest.

Alfabetul este compus din literele mari (A-Z), mici (a-z), cifrele zecimale (0-9) și următoarele caractere speciale: + - * / = < > [] . , : ; _ ^ () { } # _ .

Simbolurile speciale sînt fie caractere speciale, fie grupările de cîte două caractere speciale următoare: <>, <=, >=, :=, .., fie cuvintele rezervate următoare: AND, ARRAY, BEGIN, CASE, CONST, DIV, DO, DOWNT0, ELSE, END, FILE, FOR, FUNCTION, GOTO, IF, IN, LABEL, MOD, NIL, NOT, OF, OR, PACKED, PROCEDURE, PROGRAM, RECORD, REPEAT, SET, THEN, TO, TYPE, UNTIL, VAR, WHILE, WITH.

Identificatorii se folosesc pentru a nota constante, tipuri de date, variabile, proceduri, funcții, programe și zone de înregistrări. Se admite orice lungime de identificator. Dacă apar imediat după declarația de procedură următorii identificatori forward, external, nonPascal se numesc directive și au rol în orientarea analizei sintactice (sau generarea codului obiect).

Alți identificatori sînt recunoscuți cu o semnificație standard:

- Constante: False, True, MaxInt
- Tipuri de date: Boolean, Char, Integer, Real, Text
- Variabile: Input, Output
- Funcții intrinseci: abs, arctan, chr, cos, eof, eoln, exp, ln, odd, ord, pred, round, sin, sqr, sqrt, succ, trunc
- Proceduri predefinite: dispose, get, new, pack, page, put, read, readln, reset, rewrite, unpack, write, writeln.

2.2. Exemple de programe PASCAL

În diverse forme, toată lucrarea răspunde la întrebarea „Ce se poate face în PASCAL?” înțelegîndu-se și „Cum ?” sau „Cu ce efort, efect, grad de eleganță/portabilitate/ etc. ?”.

Această secțiune prezintă doar cîteva exemple care să-l ajute pe

cititor să se acomodeze cu atmosfera din celelalte secțiuni/capitole. Se consideră cunoscute lucrările [5], [3].

Exemplul 1:

Programul COMPRIARE_DECOMPRIARE_TEXT

```

Program comprimare_decomprimare_text;
const ASCII_max =127; sucmax =128;
      dblmax =255; radac =1; FS =28;

type  cod          =0..ASCII_max;
      ascend       =1..ASCII_max;
      descnd       =1..dblmax;
      nume_fis     =packed array [1..32] of char;
var   fis_ascii:text; nume_fis_ascii:nume_fis;
      fis_compr:file of char; nume_fis_compr:nume_fis;
      fiu:array [ascend, boolean] of descnd;
      tata:array [descnd] of ascend;
      nrbiti:0..7;
      ccmprimare:boolean;
      n:descnd;
      c:char;
      k:0..dblmax;
procedure getcmd (var nume_fis_ascii,
                  nume_fis_compr:nume_fis;
                  var comprimare:boolean);
{preia o linie de comanda de forma
  "-x out_file in_file"
unde x poate fi:
  "Comprimare", caz in care nume_fis_ascii este in_file
                  iar nume_fis_compr este out_file, sau
  "Expandare",  caz in care nume_fis_ascii este out_file
                  iar nume_fis_compr este in_file }
      external;
procedure comprim(b:boolean);
begin
      k :=k * 2; if b then k := k + 1;
      nrbiti := (nrbiti+1) mod 8;
      if nrbiti = 0 then write(fis_compr, chr(k));
end;
function expand:boolean; var c:char; begin

```

```

    if nrbiti = 0 then
        begin read(fis_compr, c); k:=ord(c); end;
    nrbiti:=(nrbiti+1) mod 8;
    expand:=k > ASCII_max;
    k:= (k*2) mod 256;
end;
procedure semi_splay_reorg(fiul:descnd);
var e1, e2:boolean; unchiul, bunicul, tatal:descnd;
begin
    tatal:=tata[fiul];
    e2 := fiul=fiu[tatal, true];
    if tatal <> radac then
        begin
            bunicul:=tata[tatal];e1:=tatal<>fiu[bunicul,true];
            unchiul:=fiu[bunicul, e1];
            fiu[bunicul, e1]:=fiul;
            fiu[tatal, e2]:=unchiul;
            tata[unchiul]:=tatal;
            tata[fiul]:=bunicul;
            if bunicul <> radac then semi_splay_reorg(bunicul);
            if comprimare then comprim(not e1);
            end;
            if comprimare then comprim(e2);
        end;
end;
begin
    for n:=2 to dblmax do tata[n]:= n div 2;
    for n:=1 to maxcar do begin
        fiu[n,false]:=n+n;
        fiu[n,true ]:=fiu[n,false]+1;
    end;
    nrbiti:=0;
    getcmd (nume_fis_ascii,nume_fis_compr,comprimare);
    if comprimare then
        begin
            reset(fis_ascii,nume_fis_ascii);
            rewrite(fis_compr,nume_fis_compr);
            repeat
                read(fis_ascii,c);
                semi_splay_reorg(ord(c)+sucmax);
            until eof(fis_ascii);
            semi_splay_reorg(FS+sucmax);
            while nrbiti <> 0 do comprim(false);
            close(fis_compr);
        end;
end;

```

```

        end
    else begin
        reset(fis_compr,nume_fis_compr);
        rewrite(fis_ascii,nume_fis_ascii);
        repeat
            n:=radac;
            repeat n:=fiu[n,expand] until n>ASCII_max;
            write(fis_ascii,chr(n-sucmax));
            semi_splay_reorg(n);
        until n=FS+sucmax;
        close(fis_ascii);
    end;
end.

```

Exemplul 2:

Prezentăm trei rutine specifice sistemului de operare RSX-11M; numărul acestora ar trebui crescut oricît de mult pentru obținerea unui mediu PASCAL (sub RSX-11M) cît mai natural și eficient.

Citirea liniei de comandă

```

procedure getcmdline;
{ Citeste o linie de comanda in memorie;
se mizeaza pe faptul ca (din meritul implementarii)
este mereu disponibila o linie de comanda (via GMCR$)
daca task-ul este instalat si lansat adecvat
}
    const
        prompt='PRM>';
    begin
        if input^ <> ' ' then
            begin
                repeat
                    get(input)
                until (input^ = ' ');
                while not eoln and (input^ = ' ') do
                    get(input);
            end;
            if input^ = ' ' then write(prompt);
            if eoln then readln;
        end;
    end;

```

Următoarele proceduri fac decompimarea-comprimarea unui șir de trei caractere RADIX-50. Funcția `ascr50` presupune că i s-a transmis un triplet de astfel de caractere, dar lăsăm ca exercițiu oprirea comprimării la primul caracter non-RADIX-50.

Lucrul cu caractere RADIX-50

```

type
  word=0..177777B;
  trio=packed array [0..2] of char;
procedure r50asc(w: word; var t:trio);
const
  rad = ' ABCDEFGHIJKLMNOPQRSTUVWXYZ$.?0123456789';
  var i:integer;
procedure a(w:word);
begin if w <> 0 then begin
  a(w div 50B); i:=i-1; t[i]:=rad[w mod 50B]; end;
end;
begin
  i:=3; a(w); while i <> 0 do begin
    i:=i-1; t[i]:=chr(0); end;
  end;
function ascr50(var t:trio):word;
var i: integer;
begin
  for i:=0 to 2 do
    if t[i] in ['a'..'z'] then
      t[i]:=chr(ord(t[i])-40B);
      ascr50:=50B*(50B*ord(t[0])+ord(t[1]))+ord(t[2]);
end;

```

2.3. O bibliotecă de grafică în PASCAL

Utilizarea rutinelor din biblioteca DAFPAS.OLB V1.1 asigură o interfață eficientă între programatorul PASCAL (Oregon 2.0H, implementat sub sistemul de operare RSX-11M) și cerințele dispozitivelor grafice din clasa DAF2020. În această secțiune nu se descrie utilizarea unor puncte de intrare existente în bibliotecă, dar care au rol intern. Biblioteca DAFPAS.OLB V1.1 face parte dintr-un grup de biblioteci care se adresează (separat) unor echipamente diverse (DAF2020, VDT52S, DGF1712) și/sau programatori în limbaje diferite (F77, C, PASCAL), dar care au comună (sau foarte asemănătoare) sintaxa majorității punctelor de

intrare. Prezentăm mai jos — pentru o anumită clasă de utilizatori — fișierul DAFCOM.pas, utilizat ca prefix (via `include dafcom.pas`) la compilarea majorității procedurilor din biblioteca DAFPAS.OLB.

```
{ $nomain } Program DAFCOM; { $own } { /* DRWCOM */ }
var X_min, Y_min, {coord vf stg jos}
    X_max, Y_max, {si dr sus pt fereastră standard}
    s_x, s_y, {factori de scala}
    c_x, c_y, {coord curente}
    c_r, s_r, {matricea de rotatie a axelor, concentrata}
    x_s, y_s, s_s, {dimensiunile textului generat}
    c_t, s_t:real; {matr rot a axei textului generat}
    line_t:integer; {tipul de linie curent}
type point=array [0..1] of real;
type segm =array [0..1] of point;
type arc  =array [0..5] of real;

{ /*      DAFCOM      */ }

var U_min, V_min,
    U_max, V_max:integer; {coord-in pixeli-ale
vizorului}
    Q_W_V:real; {factor de suprapunere
fereastră-vizor}
type string=packed array [0..7] of char;
var b_u_f:string;

procedure XMIT(n:integer;var buf:string); external;
procedure RECV(n:integer;var buf:string); external;
procedure PIXEL(x_pixel,y_pixel:integer); external;
procedure SETWND(xmin,ymin,xmax,ymax:real); external;
function CLIPP(var p,q:segm):integer; external;
function INPLG(var p:point;
               n:integer; w:point):integer; external;
function INSEG(var p:point; u:arc):integer; external;
procedure DAFINI(mode:char); external;
procedure CHANGE(mode:char); external;
procedure TYPLIN(tl:integer); external;
procedure SETVP(umin,vmin,umax,vmax:integer); external;
procedure CROSS(var ilin,jcol:integer; taste:char); external;
procedure SCALE(xs,ys:real); external;
procedure ROTAX(angle:real); external;
procedure MOVPEN(m:integer; x,y:real); external;
procedure ARCC3P(var u,v,w:point); external;
```



```

procedure POLREG(xc,yc:real;n_edges:integer); external;
procedure CERCRC(radius,xc,yc:real); external;
procedure GRAPH(function f(x:real):real; a,b:real); external;
procedure WHERE(var wx,wy:real); external;
procedure WCROSS(var wx,wy:real; taste:char); external;
procedure ERASE(part,line_or_screen:char); external;
procedure SETCP(ilin,jcol:integer); external;
procedure LETCP(direction:char;units:integer); external;
procedure GETCP(var ilin,jcol:integer); external;
procedure SCROLL(inf_line,sup_line:integer); external;
procedure SELECT(display_mode:char); external;
procedure VTXt(font:char;var txt:string); external;
function ATAN2(y,x:real):real; external;
function SIGN(x:real):integer; external;
function MIN(x,y:real):real; external;
function MAX(x,y:real):real; external;
function DET(var x,a,b:point):real; external;
procedure COPY(size:char); external;
procedure HCOPY; external;
procedure NCOPY; external;
procedure DAFEND; external;

```

Datorită acestei liste și prezentărilor din 1.3 și [6], se recomandă utilizatorului unele comparații (vezi și 1.2), desigur pentru evaluări personale.

2.3.1. Definirea sesiunii de desen

Avînd în vedere neconcordanța dintre sistemul de axe al terminalului și cel al utilizatorului precum și disponibilitățile de interactivitate, DAFPAS.OLB conține următoarele puncte de intrare:

```

DAFINI
PIXEL
WCROSS, CROSS
SETWND, SETVP
SCALE, ROTAX
TYPLIN
CHANGE
DAFEND

```

DAFINI – Inițializează modul grafic

Apelul diverselor rutine din biblioteca DAFPAS.OLB are ca efect lateral și actualizarea unor date dintre cele globale prezentate. Modulul

DAFINI inițializează toate aceste date așa cum se precizează în descrierea punctelor de intrare care le actualizează. Fereastra curentă este stabilită la dreptunghiul $[-90, 90] \times [-35, 100]$, iar vizorul curent la dreptunghiul maxim $[0, 1023] \times [0, 767]$. Prin urmare, inițial, originea axelor de coordonate se află în mijlocul ecranului.

Parametrul **mode** poate lua valorile:

- **'v'**(t100) – pentru stabilirea modului inițial de lucru cu terminalul ca video terminal de tip Vt100
- **'g'**(graphics) – ... sau de tip grafic; schimbarea ulterioară a acestui mod se face cu subrutina **CHANGE**.

Subprograme apelate : **TYPLIN**, **CHANGE**.

PIXEL – aprinde un pixel

Desenarea punctului de coordonate (**x_pixel**, **y_pixel**)
x_pixel=0,...,511 și **y_pixel**=0,...,383 (cînd terminalul este utilizat ca TK) se face cu apelul:

PIXEL(x_pixel, y_pixel);

Subprograme apelate : —

WCROSS – prela coordonate (în mm)

Apelul **WCROSS(wx, wy, taste)** conduce la afișarea cursorului cruce și la inițializarea variabilelor reale **wx** și **wy** cu valorile abscisei și ordinatei locului de afișare a cursorului (în sistemul de axe al utilizatorului), precum și a variabilei **taste** cu codul ASCII al tastei care a produs continuarea execuției.

Subprograme apelate : **CROSS**

CROSS – prela coordonate (în pixel)

Apelul **CROSS(iLin, jCol, taste)** conduce la afișarea cursorului cruce și la inițializarea variabilelor întregi **iLin** și **jCol** cu valori (în pixeli) ale liniei (0,...,1023) și coloanei (0,...,767) pe care se află cursorul (după eventuale mișcări) și a variabilei **taste** cu codul ASCII al tastei care a produs continuarea execuției.

SET WINDow – definește o fereastră

Limitarea reprezentării unor desene la porțiuni strict controlate ale universului este cerută în variate situații. Definirea acestei porțiuni ca dreptunghi cu laturile paralele cu axele de coordonate ale universului se face cu apelul:

SETWND(Xmin, Ymin, Xmax, Ymax);

parametrii fiind constante/variabile reale considerate în mm care actualizează `x_min`, `y_min`, `x_max`, `y_max`.

Subprograme apelate : —

SET ViewPort – definește vizorul

Limitarea suprafeței de reprezentare este de asemenea utilă; definirea ei se face cu apelul:

`SETVP(Umin, Vmin, Umax, Vmax);`

parametrii fiind constante/variabile întregi considerate în pixeli care actualizează `U_min`, `V_min`, `U_max`, `V_max`.

Subprograme apelate : —

ROTAX – rotația axelor utilizatorului

Stabilirea unui unghi nenul (măsurat în sens trigonometric) între noile și vechile axe utilizator (rotația axelor), se obține cu apelul:

`ROTAX(angle);`

- `angle` – variabilă/constantă reală definind mărimea (în grade sexagesimale) a unghiului de rotație, actualizându-se `c_r`, `s_r`.

După apelul `DAFINI` acest unghi este 0.

Subprograme apelate : —

SCALE – stabilirea scalelor

Unitatea utilizator de măsură a lungimii a fost stabilită, prin lipsă, milimetrul. Schimbarea ei (pentru ambele axe), se face cu apelul:

`SCALE(xs, ys);`

- `xs` – variabilă/constantă reală reprezentând lungimea în mm a noii unități de măsură a axei absciselor;
- `ys` – analog, relativ la axa ordonatelor.

După `DAFINI` `s_x` și `s_y` sînt 1.

Observație: Fiind definite fereastra și vizorul, subrutinele de reprezentare realizează automat o scalare (`Q_W_V`) care să suprapună fără deformare colțul stînga-jos al ferestrei peste cel al vizorului. Scalarea nu afectează coordonatele (în mm) ale ferestrei!

Subprograme apelate : —

TYPLIN – setează tipul de linie

Definirea modelului curent de linie se face inserînd în programul sursă linia:

`TYPLIN(tl);`

- `tl` – variabilă/constantă întreagă reprezentînd tipul de linie

utilizat:

| | |
|-------------|-----------------------------|
| $t1 < 0$ | linie întunecată |
| $t1 = 0$ | linie luminoasă continuă |
| $t1 = 1, 2$ | linie luminoasă întreruptă |
| $t1 = 3$ | linie luminoasă punctată |
| $t1 = 4$ | linie luminoasă linie-punct |
| $t1 > 4$ | este echivalent cu $t1 = 0$ |

După apelul DAFINI tipul curent de linie ($line_t$) este 0.

Subprograme apelate : —

CHANGE – schimbă modul de lucru

Schimbarea modului de lucru cu DAF-2020, în vederea utilizării de subrutine VT100 sau de grafică se face cu apelul:

CHANGE(mode);

mode fiind constantă/variabilă de tip CHAR luând valorile 'V'(t100) sau 'G'(graphics), cu semnificație evidentă.

DAFEND – sfârșitul sesiunii de desen

Sfârșitul sesiunii de desen se anunță cu apelul:

DAFEND;

care produce și selecția modului VT100.

Observație: Modulele prezentei biblioteci se apelază în interiorul unei perechi de paranteze DAFINI/DAFEND; un task poate conține oricâte astfel de perechi neconținute una în alta.

Subprograme apelate : —

2.3.2. Trasări/Ștergeri

În această secțiune se includ:

MOVPEN

care „mișcă” efectiv spotul, realizându-se un desen, precum și

COPY

utilă pentru copierea ecranului pe imprimantă CDC-9335. Pentru ștergerea unui segment se apelează TYPLIN cu parametrul -1, apoi MOVPEN pentru trecerea peste segmentul de șters (în sensul desenării). Dispozitivele DAF pot desena segmente de dreaptă (via un interpolator liniar existent hardware care unește punctele necesare de pe o rețea cu 1024×768 pixeli). Indicațiile necesare mișcării se vor constitui ca parametri ai procedurii independente de dispozitiv MOVPEN (care apelează intern HMOVE, dependentă de echipament). Ea se apelează sub forma:

MOVPEN(m, x, y);

unde

- m poate lua valorile 0, 1, 10, 11: dacă $m < 2$ nu se trasează segment; dacă $m = 0(\bmod 2)$ se vor interpreta ceilalți doi parametri ca incrementări ale coordonatelor ultimului punct desenat, și nu coordonate ale punctului curent, cum se întâmplă pentru m impar.
- x, y sînt coordonatele unui punct cu semnificația dependentă de m

Unde se mai află tocul ?

WHERE(wx, wy);

Acest apel este adesea util într-un modul care nu a stabilit el însuși acest punct curent (și eventual trebuie refăcut înainte de RETURN la chemător).

Observație: *Alte informații utile pot fi receptate din variabilele globale al căror conținut a fost descris; este lăsat în responsabilitatea utilizatorului accesul la aceste variabile (R/W neavizat) !*

COPY – copierea ecranului

Apelul COPY(size) produce copierea pe dispozitivul CDC-9335 cuplat ca hard-copy a conținutului memoriei-ecran, la dimensiunea indicată de constanta șir de caractere/taboul de elemente CHAR size, care ia valorile legale 'S' (imple)/'D' (ouble).

GETSTS – starea terminalului

Starea curentă a terminalului se află cu apelul:

GETSTS(ix, jy, cod);

inițializîndu-se ix și jy cu coordonatele (în pixeli) ale cursorului, iar octetul cod conform definițiilor din cartea tehnică (vezi ESC ENQ).

2.3.3. Desenarea textelor

Editarea textelor este facilitată de următoarele proceduri:

TEXTS

TEXTA

TEXT

Efectul apelului primelor două subrutine se menține pînă la un nou apel. Desenarea textelor ține cont de sistemul de axe al utilizatorului (scalare, rotație) și de fereastra curent definită. Subrutina VTXT produce un text exploatînd facilitățile VT100 ale terminalului, și trebuie apelată în acest mod.

TEXTS – setează dimensiunea literelor

Apelul:

TEXTS(ysize, xsize, ssize);

unde **ysize**, **xsize**, **ssize** sînt constante/variabile reale, declară înălțimea, lățimea și spațiul dintre caracterele editate ulterior, ținînd seama de scalele (pentru **Ox** și **Oy**) curente. După apelul **DAFINI** aceste dimensiuni sînt respectiv 5.(y_s), 5.(x_s), 1.(s_s)

Subprograme apelate : —

TEXTA – setează înclinarea textului

Înclinarea axei textului față de axa **Ox** a utilizatorului se face inserînd în program linia:

TEXTA(angle);

unde **angle** este variabilă/constantă reală precizînd (în grade sexagesimale) înclinarea dorită. După apelul **DAFINI** înclinarea inițială este 0.(c_t, s_t)

Subprograme apelate : —

TEXT – editarea unui text

Apelul:

TEXT(text)

unde **text** este adresa unui string, produce desenarea șirului **text**, și repoziționarea cursorului la începutul textului.

Subprograme apelate : **WHERE**, **MOVPEN**.

2.3.4. Modul VT-100

Următoarele proceduri :

ERASE

SCROLL

SETCP

SETCPS

LETCP

GETCP

SELECT

VTXT

TCOPY, **NCOPY**

facilitează exploatarea posibilităților oferite de DAF-2020 de a lucra ca terminal de tip VT100. Ele se vor apela numai după ce în prealabil s-a apelat **CHANGE**('V').

Prin utilizarea succesivă a terminalului DAF2020 ca **TEKTRONIX**

și VT100 se poate mări gradul de interactivitate al task-ului și se pot obține ecrane mixate care apoi să fie copiate pe imprimantă.

ERASE – Ștergere în ecran/linie

Față de poziția curentă a cursorului se poate șterge o porțiune de ecran sau de linie. Se realizează aceasta cu apelul:

```
ERASE(part, line_or_screen);
```

Ambii parametri sînt constante de tip șir de caractere sau tablouri de elemente CHAR.

part poate lua valorile:

- 'A'(ll) pentru ștergerea întregii entități
- 'B'(egin) pentru ștergerea entității de la început pînă în poziția curentă a cursorului
- 'E'(nd) pentru ștergerea entității de la poziția curentă a cursorului pînă la sfîrșit

line_or_screen poate lua valorile 'L'(ine) sau 'S'(creen).

Subprograme apelate : —

Defilare

După umplerea ultimei linii terminalul face defilarea în sus pentru a face loc unei noi linii (pierzînd prima linie de pe ecran).

Apelul:

```
SCROLL(inf_line, sup_line);
```

definește marginile de sus și de jos ale zonei pe care se produce defilarea. Ambii parametri sînt constante/variabile întregi (1,...,24).

Subprograme apelate : —

Poziția cursorului

Facilitate specifică terminalelor cu tub catodic, poziționarea cursorului se realizează inserînd în program linia

```
SETCP(Ilin, Jcol);
```

Ilin este constantă/variabilă întreagă cu valori de la 1 la 24; Jcol este constantă/variabilă întreagă cu valori de la 1 la 80.

Pentru poziționarea cursorului în interiorul zonei de scroll se utilizează subrutina SETCPS; cei doi parametri definesc de asemenea linia și coloana destinație a cursorului.

Pentru poziționarea diferențială a cursorului se poate utiliza subrutina LETCP cu următorii doi parametri:

- constantă/variabilă de tip CHAR cu conținutul 'U'(p) / 'D'(own) / 'L'(eft) / 'R'(ight) și cu semnificația evidentă;
- constantă/variabilă întreagă și pozitivă indicînd numărul de linii

(coloane) de deplasare a cursorului pe aceeași coloană (linie).

Subprograme apelate : —

Pentru a afla unde se află cursorul alfanumeric, se inserează în program linia:

```
GETCP(ilin, jcol);
```

cu efectul inițializării variabilelor întregi *ilin/jcol* cu valorile liniei și coloanei pe care se află cursorul după o serie de mișcări.

Selectare mod de afișare

Echipamentul poate afișa un text Normal (alb pe fond întunecat) sau Reverse video (întunecat pe fond alb); în ambele cazuri textul poate fi subliniat. Selecția stilului de afișare se obține după apelul `SELECT(display_mode)`, unde `display_mode` poate fi 'N'(ormal), 'R'(evers) sau 'U'(nderlined).

Subprograme apelate: —

Afișare text

După stabilirea poziției de început a textului și a modului de afișare, afișarea propriu-zisă se face pe calea

```
VTXT(dim, txt);
```

primul parametru poate lua valorile 'N'(ormal), 'D'(ouble) sau 'L'(arge) pentru a stabili mărimea caracterelor textului, iar al doilea conține textul de afișat.

Subprograme apelate : `GETCP`, `SETCP`.

Copiere ecran

Pentru a permite (sau a interzice) transmisia caracterelor venite de la calculator spre ecran și către imprimanta CDC-9335, se inserează în program linia

```
TCOPY;
```

sau

```
NCOPY;
```

Deoarece codurile de control ale celor două dispozitive sînt diferite, se obțin rezultate nedorite dacă se lucrează în mod transparent (`TCOPY`) și se transmit și către imprimantă coduri de control specifice ecranului.

2.3.5. Subrutine specializate

Aceste subrutine sînt de interes specializat. Setul acestor subrutine poate fi îmbogățit de fiecare utilizator, după necesitățile locale. În prezent, biblioteca DAFPAS.OLB conține subprogramele


```

INSEG, INPLG
CLIPP, USRWND, UCLIPP
LINE
GRAPH
CERCRC, POLREG
ARCC3P

```

Test de apartenență

Date fiind un punct P și un poligon cu n vîrfuri memorate în lista w de n puncte, funcția $INPLG(P, n, w)$ ia valorile:

- -1 dacă P este exterior poligonului w
- 0 dacă P se află pe frontiera poligonului
- +1 dacă P este interior;

Dacă u este o listă de 3 puncte cu semnificația punct inițial, punct intermediar și final, funcția $INSEG(P, u)$ returnează valorile -1, 0, +1 în conformitate cu poziția punctului P față de segmentul de cerc precizat de u .

Decupare

Presupunînd definită o fereastră ca poligon (convex) cu max 6 colțuri, apelul $k := UCLIPP(P, Q)$; produce codul de retur k (variabilă întregă) și coordonatele subsegmentului vizibil Q al segmentului P (liste de două variabile de tip punct).

Codul de retur k are semnificațiile următoare:

- 0 segmentul P complet vizibil
- 4 segmentul P invizibil
- 1 vizibil subsegm $Q[1]P[2]$
- 2 vizibil subsegm $P[1]Q[2]$
- 3 vizibil subsegm $Q[1]Q[2]$

Secvența de apel pentru $USRWND$ este

```
USRWND(n, wnd);
```

primul parametru fiind constantă/variabilă întregă, indicînd prin valoarea ei absolută numărul de vîrfuri ale ferestrei; dacă este negativă se omite desenarea ferestrei. Al doilea parametru este variabila de tip listă de puncte ale poligonului wnd .

Decuparea față de fereastra standard (definită cu $SETWND$) se face cu apelul $k := CLIPP(P, Q)$; parametrii au aceeași semnificație ca mai sus.

Subprograme apelate: —

Grafice de funcții continue

Dacă a, b sînt numere reale, $a < b$ și $f: [a, b] \rightarrow \mathbb{R}$ este o funcție continuă al cărei grafic se cere desenat, acest lucru se obține pe calea

`GRAPH(f, a, b);`

Este legal și un apel de forma `GRAPH (f, b, a)` cu efectul corespunzător.

Subprograme apelate: `MOVPEN`

Desenarea cercurilor

Cerc cu rază și centru date

`CERCRC(r, xc, yc);`

produce desenarea unui cerc de rază r și centrul (xc, yc) . Toți parametrii sînt reali.

Desenarea unui poligon regulat cu n laturi înscris în cercul cu centrul în (xc, yc) și avînd un vîrf în punctul curent se face prin apelul

`POLREG(xc, yc, n);`

Subprograme apelate : `WHERE, MOVPEN`

Arc de cerc definit de 3 puncte

Dacă se dorește desenarea unui arc de cerc, cunoscînd coordonatele punctelor extreme și cele ale unui punct intermediar, se poate folosi apelul

`ARCC3P(A, B, C);`

A, B, C sînt punctele inițial, intermediar și final care definesc arcul de cerc dorit. Punctul intermediar se consideră apartenent arcului.

Subprograme apelate : `MOVPEN`

2.3.6. Exemple

Scopul acestei secțiuni este să prezinte în cazuri concrete exemple de apel al subrutinelor din biblioteca `DAFPAS.OLB`; programele respective nu au nevoie de mai multe explicații decît cele din comentariile încorporate. Deși utilizează un minim de cunoștințe și/sau modele matematice, scopul principal rămîne exemplificarea modului de folosire a subrutinelor din biblioteca `DAFPAS.OLB` pentru obținerea unui desen.

Programul PĂLĂRIE

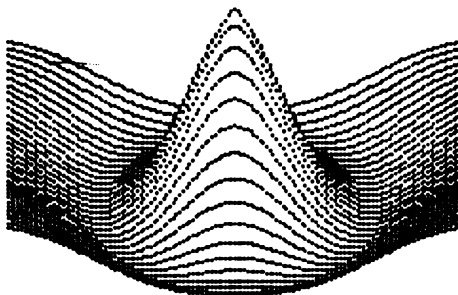
`Program Palarie;`

`{`
 desenarea pe xOz a proiecției funcției

$z = 4y + \cos(r) * \exp(-r/3)$, unde $r = \sqrt{x^2 + 16y^2} / 15$
 $-256 < x < 255$, $-16 < y < 15$

Autor: A. Posea

}



```
var r:real;
    x, y, z:integer;
var { utile simulării liniilor ascunse}
    zmin, zmax:integer;
function MIN(x, y:integer):integer;
begin
    if x<y then MIN:=x else MIN:=y;
end;
function MAX(x, y:integer):integer;
begin
    if x<y then MAX:=y else MAX:=x;
end;
procedure DAFINI(mode:char); external;
procedure PIXEL(i,j:integer); external;
procedure DAFEND; external;
begin
    DAFINI('G'); { intrare in modul Grafic}
    for x:= 180 to 355 do
        begin
            zmin:= 1023; zmax:=0;
            for y:=40 to 72 do
                begin
                    r:=(x-256)*(x-256)+16.0*(y-56)*(y-56);
                    r:=sqrt(r)/15.0;
                    z:=trunc(4.0*y+90*cos(r)*exp(-0.3333*r));
                    if (z<zmin) or (z>zmax) then PIXEL(x, z);
                    zmin := MIN(z, zmin);
```

```

        zmax := MAX(z, zmax);
    end;
end;
DAFEND;
end.

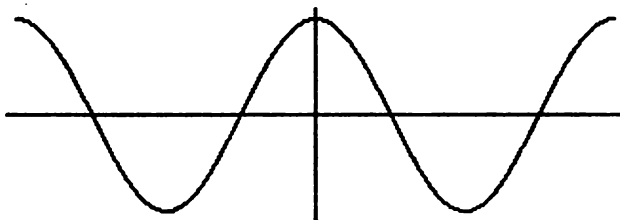
```

Programul GRAFIC

```

Program Grafic;
{
    Graficul functiei cos:[A, B]-> [-1., 1.]
    Autor: A. Posea
}

```



```

var A, B:real;
procedure dafini(mode:char); external;
procedure scale(sx, sy:real); external;
procedure movpen(k:integer; x,y: real); external;
procedure graph(function f(x:real):real; a,b: real);
    external;
procedure dafend; external;
function f(x:real):real; begin f:=cos(x) end;
begin
    A := -2.0 * 3.141592;
    B := -A;
    DAFINI('G');
    SCALE(10.0, 10.0);
    movpen(1, A, 0.0); movpen(11, B, 0.0);
    movpen(1, 0.0, 1.1); movpen(11, 0.0, -1.1);
    GRAPH(f, A, B);
    DAFEND;
end.

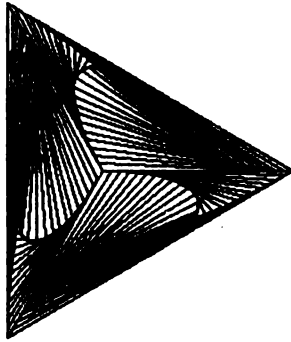
```

Programul TRIUNGHI

```

Program Triunghi;
  { Autor: A. Posea }
const
  pi=3.141592;
  lt=30.0;
type
  punct=
    record
      x:real;
      y:real;
    end;
  var A, B, C, D, E, F:punct;
      p:real;
  procedure dafini(mode:char); external;
  procedure movpen(m:integer;x,y:real); external;
  procedure copy(mode:char); external;
  procedure dafend; external;

```



```

function dist(var A, B:punct):real;
begin
  dist:=sqrt(sqr(A.x-B.x)+sqr(A.y-B.y));
end;
procedure alt_triunghi(var A, B, C:punct);
begin
  movpen(01,A.x,A.y);
  movpen(11,B.x,B.y);
  movpen(11,C.x,C.y);
  movpen(11,A.x,A.y);
end;

```

```

begin write('Introduceti 0<p<1 '); read(p);
D.x:=lt; D.y:=0.0;
E.x:=lt * cos(2.0 * pi / 3.0);
E.y:= lt * sin(2.0 * pi/3.0);
F.x:=E.x; F.y:=-E.y;
dafini('G');
repeat begin
    A.x:=D.x; A.y:=D.y;
    B.x:=E.x; B.y:=E.y;
    C.x:=F.x; C.y:=F.y;
    alt_triunghi(A, B, C);
    D.x:=A.x + p * (B.x-A.x); D.y:=A.y+p*(B.y-A.y);
    E.x:=B.x + p * (C.x-B.x); E.y:=B.y+p*(C.y-B.y);
    F.x:=C.x + p * (A.x-C.x); F.y:=C.y+p*(A.y-C.y);
end
until dist(A, D) < 0.1;
copy('S'); {copie simpla pe CDC-9335}
dafend;
end.

```

2.3.7. Probleme propuse

- 1) Se consideră programul Pălărie.
 - să se execute
 - să se rescrie înlocuind cos cu sin, apoi utilizând indicația că următoarele două cicluri realizează aceeași listă, dar al doilea (mult) mai rapid:
 - a) for x:=m to n do writeln(a*x*x+b*x+c);
 - b) p:=m*(m*a+b)+c; q:=a+a; r:=a+b;

for x:=m to n do begin writeln(p); p:=p+q*x+r;

end;
- 2) Să se conceapă structurile de date comod de utilizat pentru punct, dreaptă, cerc, segment de dreaptă, plan.

2.3.8. Observații complementare

Obținerea unui task

Biblioteca DAFPAS.OLB este utilă programatorilor în PASCAL 2.0H, implementat sub RSX-11M; rescrierea ei pentru alte compilatoare PASCAL este o opțiune realizată.

Linia de comandă pentru TKB va conține această bibliotecă, înaintea celei cu modulele OTS invocate de compilatorul PASCAL, pentru a satisface algoritmul de tratare a simbolurilor referite și definite,

Prin urmare, obținerea de task-uri din exemplele de mai sus se face pe calea:

```

>TKB
TKB>EX01/FP/CP=EX01,LB:[1,1]DAFPAS/LB,PASLIB/LB
TKB>/
Enter options:
TKB>...
TKB>//

```

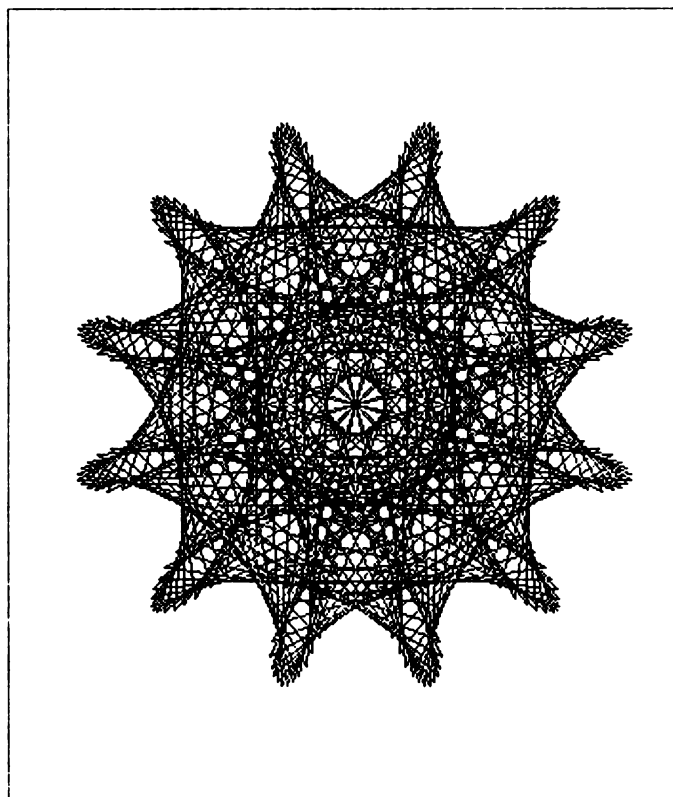
Utilizarea procedurii XMIT

Toate procedurile bibliotecii DAFPAS fac apel la serviciile procedurii XMIT, care transmite pe .MOLUN caracterele din buferul-argument. S-a ales această soluție din motivele prezentate în capitolul 1, deși existau și alte soluții (vezi secțiunea 5.7.4)

2.4. Bibliografie

-
- [1] James FOLEY, Andries Van DAM
Fundamentals of interactive computer graphics, Addison-Wesley Publ Co, 1981
 - [2] A T BERZTIS
Data Structures – Theory and Practice, Academic Press, 1971
 - [3] I. VĂDUVA, T. BALANESCU, S. GAVRILĂ, H. GEORGESCU, M. GHEORGHE, I. SOFONEA
Concepție modernă de programare în limbajul PASCAL, Tipografia Universității din București, ed II, 1989
 - [4] T. GEBER, I. MIU, V. CRISTEA, R. BULGAKOV, V. SAVESCU, M. VUICI
Echipamente periferice, vol 3, Editura Tehnică, București 1986
 - [5] Peter GROGONO
Programming in PASCAL, Addison-Wesley Series in Computer Science, 1978
 - [6] M. VLADA, A. POSEA
Grafică automată în limbajul FORTRAN77 și aplicații, Tipografia Universității din București, Ediția a II^a, 1990

GRAFICĂ ÎN LIMBAJUL TURBO PASCAL



3. GRAFICĂ ÎN LIMBAJUL TURBO PASCAL 87

| | |
|---|-----|
| 3.1. Sistemul de operare MS-DOS | 89 |
| 3.2. Mediul de programare TURBO PASCAL | 91 |
| 3.2.1. Operarea în TURBO PASCAL | 92 |
| 3.2.2. Memorator pentru utilizarea limbajului TURBO PASCAL | 94 |
| 3.2.2.1. Declarații | 94 |
| 3.2.2.2. Comenzi | 103 |
| 3.2.3. Programe în limbajul TURBO PASCAL. | 108 |
| 3.3. Biblioteca grafică TURBO PASCAL | 112 |
| 3.3.1. Prezentare generală | 112 |
| 3.3.2. Definirea sesiunii de desen | 113 |
| 3.3.2.1. Inițializarea unei sesiuni de desen | 113 |
| 3.3.2.2. Detectarea erorilor | 116 |
| 3.3.2.3. Coordonate | 116 |
| 3.3.2.4. Fixare vizor(viewport). Decupare(clipping) | 117 |
| 3.3.2.5. Punct curent | 118 |
| 3.3.2.6. Scalarea și rotația | 119 |
| 3.3.2.7. Fixarea culorii | 120 |
| 3.3.2.8. Închiderea sesiunii de desen | 122 |
| 3.3.3. Desenarea textelor | 122 |
| 3.3.3.1. Setări pentru trasarea caracterelor | 122 |
| 3.3.3.2. Trasarea unui text | 123 |
| 3.3.3.3. Trasări numerice | 123 |
| 3.3.4. Trasări | 124 |
| 3.3.4.1. Trasări de segmente | 124 |
| 3.3.4.2. Trasări de dreptunghiuri, poligoane, cercuri, arce de cerc și elipse | 126 |
| 3.3.4.3. Hașuri | 127 |
| 3.3.5. Rutine la nivel de pixel | 129 |
| 3.3.6. Diverse | 130 |
| 3.3.7. Ștergerea și copierea ecranului | 131 |
| 3.3.8. Constante, tipuri de date și variabile definite în biblioteca grafică GRAPH.TPU | 132 |
| 3.3.9. Program demonstrativ — Exemple de utilizare | 136 |
| 3.3.10. Algoritmi și programe de grafică | 150 |
| 3.4. Probleme propuse | 166 |
| 3.5. Anexe | 168 |
| 3.5.1. Lista comenzilor MS-DOS V4.0 | 168 |
| 3.5.2. Cuvinte rezervate în TURBO PASCAL versiunea 5.5 ... | 170 |
| 3.5.3. Proceduri și funcții în biblioteca grafică GRAPH.TPU . | 171 |
| 3.6. Bibliografie | 172 |

3.1. Sistemul de operare MS-DOS

MS-DOS este un sistem de operare destinat gestionării resurselor software și hardware ale microcalculatoarelor compatibile cu IBM-PC. Sistemul MS-DOS este implementat pe sistemele IBM-PC (AT și XT), IBM-PS/2, ROBOTRON EC-1834, HEWLETT PACKARD, SANYO 650, OLIVETTI M24, etc.

Sistemul MS-DOS a fost implementat pe calculatoarele românești Felix-PC, Junior-PC. Felix-PC este compatibil IBM și este produs de Întreprinderea de Calculatoare Electronice București. Acesta are o structură modulară, fiind alcătuit dintr-un modul de bază și din module de extensie.

Modulul de bază conține următoarele resurse:

- unitate de prelucrare bazată pe microprocesoarele 8086 și 8087;
- memorie RAM de 256 Ko;
- memorie EPROM de 8 Ko-64Ko;
- cuplor pentru discuri flexibile de 5¼" sau 8";
- interfețe pentru:
 - tastatură
 - imprimantă serială
 - comunicație asincronă-sincronă
 - casetă magnetică audio
- generator de tonuri;
- ceas de timp real;
- număratoare programabile;
- sistem de întreruperi;
- canale de acces direct la memorie;
- conectori pentru module de extensie;
- conectori pentru periferice.

Sistemul Felix-PC poate fi prevăzut cu microprocesorul matematic NDP 8087 care crește considerabil viteza de lucru în cazul prelucrărilor numerice. Discurile flexibile de 5¼" au o capacitate de 360 Ko sau 720 Ko.

Configurația hardware minimă necesară funcționării sistemului MS-DOS este:

- procesor 16/32 biți;
- memorie internă minimă 128 Ko;
- o unitate disc flexibil;
- consolă.

Pe lângă acestea se mai pot cupla:

- unități de discuri flexibile;
- unități de discuri Winchester (10, 20, 40, 80, ...Mo);
- imprimantă;
- plotter;
- scanner;
- digitizor;
- extensii de memorie.

Sub sistemul de operare MS-DOS se pot executa programe de cele mai diferite tipuri:

- compilatoare (FORTRAN, FORTRAN-77, PASCAL, TURBO PASCAL, C, C++, TURBO C, COBOL, TURBO PROLOG, LISP86, MODULA 2, ADA)
- interpretoare (BASIC, GWBASIC);
- asamblatoare (TASM, MASM);
- depanator (DEBUG);
- editor de legături (LINK);
- bibliotecar (LIB);
- editoare de text (WORDSTAR, WORDPERFECT, EDIT, EDLIN);
- sisteme de gestiune a bazelor de date (dBASE, FOXPRO, PARADOX, ORACLE);
- programe de comunicații (KERMIT);
- utilitare (NORTON COMMANDER, SPY, EXPLORER, PC TOOLS);
- generatoare de rapoarte, tabele (MULTIPLAN, LOTUS, QUATTRO);
- programe pentru tehnoredactare (VENTURA PUBLISHER);
- programe de desen (AUTOCAD);
- soft științific (MATHLAB, MATHCAD, EUREKA, REDUCE);
- programe de pictură (PC PAINTBRUSH);
- programe antivirus (SCAN, TNTVIRUS).

Lansarea sistemului MS-DOS începe cu execuția componentei BIOS (Basic Input Output System) care testează funcționarea modulelor

calculatorului și inițializează o parte din acestea. Această componentă se lansează automat când se conectează microcalculatorul la rețeaua electrică.

BIOS-ul predă controlul încărcătorului sistemului, program care încarcă fișierele BIO.COM și DOS.COM de pe discul sistem. Aceste fișiere sînt „invizibile”, ele nu pot fi citite, copiate sau executate prin comenzile obișnuite.

Lansarea sistemului execută și operația de configurare a sistemului (fixarea numărului de fișiere deschise simultan, număr buffere, etc), care depinde de prezența fișierului CONFIG.SYS în directorul rădăcină al discului sistem.

După încărcarea MS-DOS-ului, sistemul caută fișierul COMMAND.COM, îl încarcă și îl lansează în execuție. Acesta este interpretorul de comenzi care acționează ca o interfață între utilizator și sistemul de operare. Interpretorul caută fișierul de comenzi AUTOEXEC.BAT. Dacă îl găsește, execută comenzile conținute în acest fișier, altfel lansează un dialog pentru introducerea datei și orei curente, după care afișează prompterul sistem (format din numele unității implicite și caracterul >).

Din acest moment se poate lansa o comandă MS-DOS (vezi anexa 3.5.1) sau se poate lansa în execuție un program.

Pentru a putea fi gestionate, unitățile de disc (floppy și hard) au fost etichetate (de obicei, A și B pentru floppy disk și C pentru hard disk).

Pe o unitate de disc fișierele pot fi grupate în directoare (cataloge) după anumite criterii stabilite de utilizator. Fișierele unui director pot fi grupate în continuare, în subdirectoare, obținîndu-se astfel o structură arborescentă. Un fișier este unic determinat de numele său, extensia sa și calea în care se găsește în structura arborescentă a directoarelor.

Informații suplimentare legate de sistemul de operare MS-DOS se găsesc în [1] și [2].

3.2. Mediul de programare TURBO PASCAL

În sistemul clasic, programatorul avea la dispoziție un editor propriu pentru a crea și modifica programele sursă. Acestea se compilau cu ajutorul compilatorului pentru limbajul respectiv, indicîndu-se în linia de comandă numele fișierului și o serie de opțiuni de compilare. Modulele obiect erau apoi linkeditate obținîndu-se programul executabil. Acest mod de lucru este accesibil în TURBO PASCAL cu condiția ca pe discul utilizator să se găsească fișierul TPC.EXE.

Ca o noutate, TURBO PASCAL-ul oferă un mediu integrat de realizare și execuție a programelor care combină facilitățile de editare, de lucru cu fișiere, de compilare, de linkeditare și execuție a unui program. Programatorul are la dispoziție meniuri; ferestre, controlul configurației,

help-uri. Pentru acest mod de lucru, pe discul utilizator trebuie să se afle fișierul TURBO.EXE.

Cu ajutorul programului de instalare TINST.EXE se pot schimba diferite valori implicite ale mediului de programare TURBO PASCAL, cum ar fi: mărimea ecranului de afișare, comenzi de editare, culorile meniului și directoarele implicite.

3.2.1. Operarea în TURBO PASCAL

Lucrul cu mediul de programare TURBO PASCAL presupune existența pe discul utilizatorului a fișierelor TURBO.EXE (mediu integrat de realizare și execuție a programelor), TURBO.TPL (biblioteci rezidente — dacă se apelează) și TINST.EXE.

Mediul TURBO PASCAL dispune de un editor, un compilator și un linkeditor încorporat. TURBO PASCAL-ul se lansează cu comanda:

C:\>TURBO

presupunând că fișierul TURBO.EXE se găsește pe unitatea C, în directorul rădăcină.

În acest moment, mediul de realizare și execuție a programelor TURBO PASCAL este la dispoziția utilizatorului.

Pe ecranul terminalului se afișează meniul principal:

| | | | | | | |
|------|------|---------|-----|---------|-------|-------------|
| File | Edit | Compile | Run | Options | Debug | Break/watch |
|------|------|---------|-----|---------|-------|-------------|

Selecția unui submeniu din meniul principal se realizează tastând simultan tasta ALT și tasta care reprezintă prima literă din numele submeniului ales. Implicit sistemul se fixează asupra submeniului Edit. Utilizatorul poate să creeze un program sursă nou sau să modifice un program sursă creat anterior. Programul asupra căruia acționează utilizatorul este afișat pe ecran și devine program sursă curent.

Programul sursă curent poate fi corectat, compilat, executat, depanat, reexecutat și salvat pe disc.

- Comanda Alt+F va afișa pe ecran submeniul legat de lucrul cu fișiere din care, utilizatorul își va alege funcția dorită:

| | |
|-----------|--------------------------------------|
| LOAD (F3) | încarcă un program sursă de pe disc |
| PICK | alege un program sursă dintr-o listă |
| NEW | se creează un nou program sursă |
| SAVE (F2) | salvează programul sursă curent |
| WRITE TO | scrie programul curent sub un nume |

| | |
|------------|---|
| DIRECTORY | afișează conținutul directorului în care se lucrează |
| CHANGE DIR | permite schimbarea directorului sau a unității de disc |
| DOS SHELL | temporar se trece în MS-DOS din TURBO PASCAL |
| QUIT | se iese din mediul de programare TURBO PASCAL în MS-DOS |

- Comanda Alt+E va disponibiliza comenzile de editare asupra programului curent. Comenzile sînt asemănătoare celor din editoarele de texte WORDSTAR și EDIT.
- Comanda Alt+C permite afișarea submeniului pentru compilare.
- Comanda Alt+R afișează submeniul pentru executarea programului curent.
- Comanda Alt+O permite afișarea submeniului pentru setarea parametrilor de lucru ai mediului de realizare și execuție a programelor.
- Comanda Alt+D afișează submeniul pentru depanarea programelor.
- Comanda Alt+B vizualizează un submeniu care permite afișarea conținutului unor variabile și structura datelor în timp ce programul se execută cu depanatorul mediului de programare.

Pentru a alege și executa o funcție dintr-un submeniu se tastează prima literă din denumirea funcției sau se folosesc tastele ↑ și ↓ urmate de ENTER.

Documentația generală a limbajului sau descrierea detaliată a unei funcții sau proceduri se obține apăsînd concomitent tastele CTRL și F1.

Tasta ESC realizează trecerea din submeniu în meniul principal.

CUM SE REALIZEAZĂ UN PROGRAM ÎN TURBO PASCAL ?

Pentru început se tastează comanda:

C:\>turbo

Pe ecran va apărea meniul principal.

Se tastează Alt+F și se selectează subcomanda NEW pentru a crea un nou program sursă TURBO PASCAL. Implicit, mediul de programare disponibilizează funcțiile editorului, permițînd introducerea programului sursă TURBO PASCAL, care devine program sursă curent.

După ce a fost introdus, programul sursă curent poate fi salvat pe

disc sau poate fi executat.

Cu ajutorul comenzii Alt+R se selectează submeniul pentru execuție. Din acest submeniu se alege funcția Run (se tastează R) pentru a executa programul. Programul este mai întâi compilat. Dacă apar erori în compilare, cursorul se fixează la prima eroare de compilare găsită, afișându-se și un mesaj explicativ. Dacă nu există erori în compilare, programul este link-editat și apoi se execută.

Salvarea pe disc, în directorul curent, a programului sursă curent se realizează astfel:

- se tastează Alt+F pentru a selecta submeniul legat de lucrul cu fișiere;
- se selectează funcția WRITE TO dacă programul este nou sau se dorește salvarea sub un alt nume;
- se selectează funcția SAVE dacă se salvează sub același nume, un program sursă curent care a fost creat prin citirea de pe disc.

Lanțul de operații necesar pentru a corecta sau îmbunătăți un program creat anterior este următorul:

- se tastează Alt+F;
- se selectează subcomanda LOAD;
- se indică numele programului sursă care se dorește a fi modificat;
- se modifică, se compilează și se execută programul în modalitatea descrisă mai sus;
- se salvează programul tastând Alt+F și alegând dintre funcțiile submeniului pe SAVE. Vechea versiune a programului se păstrează cu extensia .BAK, iar versiunea actuală cu extensia .PAS.

3.2.2. Memorator pentru utilizarea limbajului TURBO PASCAL

Formal, un program scris în TURBO PASCAL are următoarea structură:

```
PROGRAM nume_program (parametri_program);  
    declarații  
BEGIN  
    comenzi  
END.
```

3.2.2.1. Declarații

Obiectele prelucrate de programele TURBO PASCAL sînt cele definite prin tipurile de date permise de limbaj. Obiectele simple, atomice sînt cele corespunzătoare tipurilor simple. Un tip structurat descrie o mulțime de obiecte identic structurate. Aceste obiecte pot fi invocate într-un program atît prin intermediul constantelor cît și al variabilelor. În

primul caz, un același obiect se asociază unic unei constante, în al doilea caz, se asociază la diverse obiecte de tipul variabilei.

În TURBO PASCAL, declarațiile constituie o parte din program ce conține pînă la 6 secțiuni de una din formele:

| | |
|--------------------------------------|--------------|
| ● declararea bibliotecilor utilizate | USES... |
| ● declararea de etichete | LABEL... |
| ● declararea de constante | CONST... |
| ● declararea de tipuri de date | TYPE... |
| ● declararea de variabile | VAR... |
| ● declararea de proceduri | PROCEDURE... |
| ● declararea de funcții | FUNCTION... |

DECLARAREA BIBLIOTECILOR UTILIZATE (UNIT-uri)

Un program poate apela proceduri și funcții definite într-o bibliotecă de programe obiect. Biblioteca de programe obiect folosită trebuie declarată cu ajutorul instrucțiunii USES:

```
USES nume_bibliotecă_1, ..., nume_bibliotecă_n;
```

Tipurile de date, structurile de date definite în aceste biblioteci pot fi accesate de programul utilizator. Cele mai utilizate biblioteci de programe sînt: CRT, SYSTEM, GRAPH, GRAPH3, DOS, OVERLAY. Utilizatorul poate să-și construiască propriile biblioteci. Pentru a afla modul de realizare a bibliotecilor se poate consulta lucrarea [3].

Procedurile și funcțiile unit-ului SYSTEM pot fi apelate chiar dacă acesta nu a fost declarat în instrucțiunea USES.

DECLARAREA DE ETICHETE

O declarație de etichete are forma:

```
LABEL et1, et2, et3, ..., etn;
```

unde et1, et2, et3, ..., etn sînt numere sau identificatori. Identificatorii sînt alcătuiți dintr-o literă sau ' ' (liniuță de subliniere) urmată de orice combinație de litere, cifre sau '_'. Lungimea identificatorilor este limitată la 127 caractere.

Declarația etichetei este valabilă numai în cadrul blocului de definiție.

DECLARAREA DE CONSTANTE

În TURBO PASCAL constantele pot primi nume simbolice (identificatori). Referirea în program la un astfel de nume este echivalentă cu utilizarea constantei asociate.

O definiție de constante simbolice are forma:

```
CONST identificator = constantă ;
```

Toate obiectele simple pot fi descrise prin constante. Exemple:

- constante întregi: 0 -220 355
- constante reale: 0.0 0.0271 -876.23 70.2e-3
- constante caracter: 'a' 'A' '?' '1'
- constante logice: TRUE FALSE

Pentru mulțimi de obiecte ordinale se pot folosi constante-mulțime cu elemente constructori de forma:

['A', 'P'] ['C'... 'N', 'X']

Constantele simbolice nu pot fi modificate în interiorul programului. Limbajul TURBO PASCAL permite definirea unor constante cu tip care sînt de fapt variabile inițializate.

Se admit constante de tip tablou, înregistrare, mulțime și expresie constantă. Sintaxa pentru declararea acestor constante cu tip este:

- tablouri
 identificator : tip = (listă de componente)
- înregistrări
 identificator : tip = (selector_1:valoare_1;

 selector_n:valoare_n)
- mulțimi
 identificator : tip = [...]
- expresie constantă
 identificator : tip = expresie_constantă

DEFINIȚII DE TIP

În TURBO PASCAL tipurile se pot clasifica în :

- tipuri predefinite, standard
- tipuri definite de utilizator

sau în :

- tipuri simple sau scalare
- tipuri structurate.

Tipurile standard întreg, real, caracter și logic sînt și tipuri simple. Alt tip standard este tipul text, dar acesta este un tip structurat.

Tipurile definite de utilizator pot fi de următoarele categorii: enumerat și subdomeniu, ca tipuri simple și cu structuri de tablou, înregistrare, mulțime și fișier, ca tipuri structurate. O categorie deosebită de tipuri care ar putea fi asimilate cu tipurile simple, în sensul de tipuri nestructurate, sînt tipurile referință. Ele sînt tipuri definite de utilizator.

Tipurile ordinale sînt tipuri simple în cadrul cărora valorile sînt strict ordonate. În această categorie intră tipurile întreg, caracter, logic și enumerat, ca și subdomeniile lor.

Tipurile predefinite sînt:

| Cuvînt rezervat | Domeniu de valori | Reprezentare în memorie |
|-----------------|---------------------------|-------------------------|
| INTEGER | -32768...32767 | 16 biți |
| WORD | 0...65535 | 16 biți |
| CHAR | Caracterele ASCII extinse | 1 octet |
| BOOLEAN | FALSE, TRUE | 1 octet |
| BYTE | 0...255 | 8 biți |
| SHORTINT | -128...127 | 8 biți |
| LONGINT | -2147483648...2147483647 | 32 biți |
| REAL | 2.9E-45...1.7E+38 | 6 octeți |
| SINGLE | 1.5E-45...3.4E+38 | 4 octeți |
| DOUBLE | 5.E-324...1.7E+308 | 8 octeți |
| EXTENDED | 1.9E-4951...1.1E+4932 | 10 octeți |
| COMP | -2E+63+1...2E+63-1 | 8 octeți |
| TEXT | 1...255 caractere | 1...255 octeți |

Tipurile definite de utilizator se declară astfel:

```
tipuri subdomeniu = const_ordinal..const_ordinal
tipuri enumerate  = (id1, id2,..., idm)
tipuri interval   = valoare_inițială..valoare_finală
tipuri tablou     = ARRAY[tipuri_de_indexare] OF tip
tipuri înregistrare simplă =
```

```
    = RECORD
        id1 : tip ;
        ...
        idm : tip ;
    END
```

sau

```
    = RECORD
        id1 : tip ;
        ...
        idm : tip ;
    CASE tip OF
        ctl : (listă_cîmpuri) ;
```

```

...
ctj : (listă_cimpuri) ;
END
tipuri mulțime = SET OF tip
tipuri fișier = FILE OF tip
tipuri referință = ^tip
    Specificațiile de tip pot să apară fie în definirea unor noi tipuri la
    introducerea de noi identificatori de tip:
        TYPE identificator = tip ;
    fie în declararea de variabile :
        VAR listă_variabile : tip ;

```

TIPUL STRING

Este un tip structurat, asemănător tabloului de caractere cu deosebirea că într-un șir, numărul de caractere poate varia dinamic între 0 și o limită superioară specificată.

Un șir se declară astfel:

```
identificator_șir = STRING[întreg]
```

unde $0 \leq \text{întreg} \leq 255$. Șirurile ocupă în memorie un număr de octeți egal cu lungimea maximă a șirurilor plus 1.

Șirurile sînt manipulate folosind expresii cu șiruri care conțin constante șir, variabile șir, apeluri de funcții cu valoare șir și operatori. Șirurile pot fi concatenate.

Dacă la atribuirea unui șir către o variabilă_șir, lungimea maximă a variabilei_șir este depășită de șir, caracterele în plus sînt trunchiate.

Proceduri standard pentru șiruri:

| | |
|--------------------------|--|
| DELETE(șir, poz, nr) | șterge nr caractere din șir începînd cu poziția poz |
| INSERT(șir1, șir2, poz) | se inserează șir1 în șir2 începînd din poziția poz |
| STR(valoare, șir) | convertește valoare (întreagă sau reală) în șir de caractere; |
| VAL(șir, variabilă, cod) | convertește un șir într-o valoare întreagă sau reală după tipul variabilei și o depune în variabilă. cod este o variabilă cu valoarea 0 dacă conversia s-a efectuat în bune condiții. Altfel, cod conține poziția în care s-a detectat eroarea în conversie. |

Funcții standard pentru șiruri:

| | |
|----------------------------------|---|
| COPY (șir, poz, nr) | extrage din șir un subșir de nr caractere începînd cu poziția poz |
| CONCAT (șir_1, șir_2,...) | întoarce un șir format din concatenarea șirurilor argument; |
| LENGHT (șir) | întoarce lungimea șirului șir |
| POS (obiect, țintă) | caută prima apariție a lui obiect în țintă. |

Operații cu mulțimi

| | |
|----|-------------|
| * | intersecție |
| + | reuniune |
| - | diferență |
| = | egalitate |
| <> | inegalitate |
| >= | include pe |
| <= | inclusă în |
| IN | apartenență |

DECLARAREA DE VARIABLE

Domeniul de valabilitate al variabilelor se definește textual, ca zonă din program în care acționează declarația unei variabile. Această zonă este de obicei întreg textul programului în care a fost declarată variabila, începînd din punctul declarației, inclusiv subprogramele declarate în interiorul (sub)programului, în afara celor în care numele variabilei a fost redeclarat.

Deci, în TURBO PASCAL avem programe cu structura de bloc, în care blocurile sînt corpurile subprogramelor. Structura permite ca domeniile de valabilitate a două variabile să fie ori disjuncte, ori incluse unul în celălalt.

Variabilele pot fi de două tipuri: statice și dinamice.

Variabilele statice „trăiesc” cît timp se execută subprogramele sau comenzile aparținînd interiorului domeniului lor de valabilitate. Variabilele dinamice au „durata de viață” dirijată prin program. Ele sînt create cu procedura **NEW** și eliberate cu procedura **DISPOSE**.

Declararea variabilelor se face în secțiunea **VAR**, astfel:

var1, var2, ..., varn: identificator_tip/descriere_tip

Oricărei variabile îi corespunde un spațiu în memorie și un nume pentru accesul la spațiul respectiv. Mărimea spațiului depinde de tipul

asociat variabilei prin declarație. Variabilele pot fi simple sau structurate.

- O componentă de tablou se reprezintă în program ca o variabilă indexată:

```
nume_tablou[listă_expresii_indiciale]
```

- O componentă de înregistrare se reprezintă în program ca o variabilă selectată:

```
nume_inregistrare.selector
```

- O componentă de fișier, cea curent accesibilă, se reprezintă în program prin variabila tampon a fișierului:

```
nume_fișier
```

DECLARAREA DE PROCEDURI ȘI FUNCȚII

Pot fi definite două tipuri de subprograme:

- funcții;
- proceduri.

Structura unei declarații de subprogram de tip funcție este:

```
FUNCTION identificador(listă_parametri) : tip ;
    declarații
BEGIN
    comenzi
END;
```

unde `listă_parametri` este o listă de specificații de forma:

```
listă_identificatori : tip
```

sau

```
VAR listă_identificatori : tip
```

separate prin caracterul `','`. Numele funcției poate apare fie printre comenzi, fie într-un apel, eventual recursiv, de funcție, de forma:

```
id_funcție(listă_de_expresii)
```

Structura unei declarații de subprogram de tip procedură este:

```
PROCEDURE identificador(listă_parametri) ;
    declarații
BEGIN
    comenzi
END;
```

Apelul unei proceduri este de forma:

```
identificador_procedură(listă_parametri_actuali);
```

Un subprogram poate transfera ca parametri alte subprograme, în care caz antetul acestora, incluzînd numele și parametrii, trebuie specificat în declarația subprogramelor.

O declarație este valabilă cît timp sîntem în blocul care conține această declarație sau într-un bloc subordonat acestuia.

FIȘIERE

Fișierele pot fi:

- de tip text (articolele sînt caractere structurate pe linii)
- cu tip (articolele sînt de același tip numit tipul de bază al fișierului)
- fără tip (articolele sînt de lungime fixă)

Proceduri standard pentru lucrul cu fișiere:

| | |
|----------------------------|--|
| ASSIGN(var_fiș, șir) | Fișierul cu numele șir aflat pe disc este asociat lui var_fiș astfel încît operațiile asupra ei vor fi efectuate asupra fișierului. |
| RESET(var_fiș) | Deschide un fișier existent. |
| REWRITE(var_fiș) | Creează și deschide un nou fișier. |
| READ(var_fiș,var1,var2,..) | Fiecare variabilă se citește din fișierul de pe discul asociat lui var_fiș, iar indicatorul fișierului avansează la componenta următoare. |
| WRITE(var_fiș,v1,v2,..) | Scrie o înregistrare în fișier. |
| SEEK(var_fiș, nr) | Poziționează indicatorul fișierului pe a nr-a componentă. |
| FLUSH(var_fiș) | Zona tampon a unui fișier deschis în scriere este golită automat (scrisă) în fișierul fizic de pe disc. Este utilă în cazul zonelor tampon mari. |
| CLOSE(var_fiș) | Închide un fișier deschis. |
| ERASE(var_fiș) | Șterge fișierul de pe discul asociat lui var_fiș. |
| RENAME(var_fiș, șir) | Dă un nou nume (șir) fișierului asociat lui var_fiș. Nu se folosește pe un fișier deschis. |

Procedurile standard GET și PUT nu sînt implementate.

Funcții standard pentru lucrul cu fișiere:

| | |
|--------------------------|---|
| FILEPOS(var_fiș) | Întoarce o valoare întreagă reprezentînd poziția curentă a indicatorului fișierului. |
| FILESIZE(var_fiș) | Întoarce o valoare întreagă reprezentînd dimensiunea fișierului. |
| EOF(var_fiș) | Întoarce o valoare logică : TRUE dacă indicatorul fișierului este poziționat la sfîrșitul acestuia, FALSE în caz contrar. |
| EOLN | Întoarce o valoare logică : TRUE dacă poziția curentă în fișier este la sfîrșitul unei linii sau la sfîrșitul fișierului, FALSE în caz contrar. |

Variabilele de tip fișier text sînt declarate cu ajutorul tipului standard **TEXT**.

Funcții specifice fișierelor de tip text:

| | |
|--------------------------|---|
| SEEKEOLN(var_fiș) | Este similar cu EOLN . Ignoră spațiile și TAB -urile înainte de a testa sfîrșitul liniei. |
| SEEKEOF(var_fiș) | Este similar cu EOF . Ignoră spațiile, TAB -urile și marcasele de sfîrșit de linie (secvențe <CR><LF>) înainte de a testa sfîrșitul fișierului. |

DISPOZITIVE LOGICE

Este permisă utilizarea dispozitivelor externe prin intermediul dispozitivelor logice, tratate ca fișiere text.

Dispozitivele logice disponibile sînt:

- **CON** : consola ; cu ecou, se preia o linie;
- **TRM** : terminal ; cu ecou, se preia un caracter;
- **KBD** : tastatura ; fără ecou, se preia un caracter;
- **LPT1, LPT2, LPT3** : imprimanta;
- **COM1, COM2** : dispozitiv extern auxiliar;
- **USR** : dispozitiv logic definit de utilizator.

3.2.2.2. Comenzi

Comenzile acoperă toate structurile de control folosite în programarea structurată. Acestea sînt de tipul:

- **SECVENȚEI:**

```
comandă ;
... ..
comandă ;
```

sau

```
BEGIN
    comandă ;
    ... ..
    comandă ;
END
```

- **DECIZIEI:**

```
IF condiție THEN
    comandă
ELSE
    comandă ;
```

sau

```
IF condiție THEN
    comandă ;
```

- **SELECTIEI:**

```
CASE expresie OF
    subdomeniul_1 : comandă ;
    ... ..
    subdomeniul_n : comandă ;
END;
```

sau

```
CASE expresie OF
    subdomeniul_1 : comandă ;
    ... ..
    subdomeniul_n : comandă ;
ELSE
    comandă ;
END;
```

- **CICLULUI CU CONDIȚIE:**

```
WHILE condiție DO
    comandă ;
```

sau

```
REPEAT
    comandă ;
    ... ..
```

```

        comandă ;
    UNTIL condiție ;
● CICLULUI CU CONTOR:
    FOR var := vi TO vf DO
        comandă ;
sau
    FOR var := vi DOWNTO vf DO
        comandă ;
Comenzile simple sînt:

```

| | |
|--------------------|----------------------------|
| ATRIBUIREA | var := expresie; |
| SALT NECONDIȚIONAT | GO TO etichetă; |
| APEL DE PROCEDURĂ | nume_procedură(parametri); |
| COMANDA VIDĂ | |

ATRIBUIREA se poate efectua asupra oricărui tip de obiecte (în afara fișierelor), cu condiția ca expresia și variabila din stînga atribuirii să fie de același tip. Excepție face atribuirea la o variabilă reală a unei expresii întregi.

Prin comanda GO TO nu se poate părăsi blocul curent, adică nu se pot face salturi în și din subprograme.

Operatorii folosiți în expresii și condiții, în ordine descrescătoare a priorităților sînt:

| | | | | | | | |
|------------|---|-----|-----|-----|-----|---|----|
| minus unar | | | | | | | |
| NOT | | | | | | | |
| * | / | DIV | MOD | AND | | | |
| SHL | | | | SHR | | | |
| + | - | OR | | | XOR | | |
| # | < | <= | = | <> | >= | > | IN |

- Operatorul NOT se aplică și întregilor rezultînd complementul față de 1 al întregului respectiv.
- AND, OR și XOR sînt, pe lîngă operații logice și operații asupra întregilor considerați drept configurații de biți.
- SHL și SHR sînt operații de deplasare la stînga, respectiv la dreapta a întregilor considerați drept configurații de biți.

INSTRUCȚIUNI DE INTRARE/IEȘIRE DIN FIȘIERELE STANDARD INPUT / OUTPUT

```

READ(listă_variabile)
READLN(listă_variabile)
WRITE(listă_variabile)
WRITELN(listă_expresii)

```

INSTRUCȚIUNI DE INTRARE/IEȘIRE CU CONVERSIE AUTOMATĂ DIN FIȘIER TEXT

```

READ(ume_fișier, listă_variabile)
READLN(ume_fișier, listă_variabile)
WRITE(ume_fișier, listă_expresii)
WRITELN(ume_fișier, listă_variabile)

```

PROCEDURI DE CREARE ȘI DISTRUGERE VARIABLE DINAMICE

NEW(variabilă de tip referință)

Creează o nouă variabilă dinamică și poziționează un pointer la începutul zonei.

DISPOSE(variabilă de tip referință)

Eliberează zona ocupată de o variabilă dinamică referită de variabila-parametru.

PROCEDURI ȘI FUNCȚII STANDARD

Proceduri:

| | |
|-------------|---|
| CLRSCR | șterge ecranul și poziționează cursorul în stînga sus; |
| CRTINIT | inițializare terminal; |
| CRTEXT | resetare terminal; |
| DELAY(ms) | realizează o înțrziere de ms milisecunde; |
| DELLINE | șterge linia curentă; |
| INSLINE | inserare linie curentă în poziția curentă; |
| GOTOXY(x,y) | poziționare cursor; |
| EXIT | realizează trecerea din blocul curent în blocul imediat superior; |

| | |
|--------------------------------|--|
| HALT | oprire execuție program și revenire în sistemul de operare; |
| RANDOMIZE | inițializare generator de numere aleatoare; |
| LOWVIDEO | poziționează caracteristica video pe intensitatea mică; |
| NORMVIDEO | selectează atributele originale ale textului; |
| MOVE (var_1, var_2, nr) | mută nr octeți începînd cu primul octet ocupat de var_1 într-o zonă ce începe cu primul octet ocupat de var_2; |
| FILL (var, nr, val) | umple o zonă de nr octeți (începînd de la primul octet ocupat de var) cu valoarea val. |

Funcții:

| | |
|-------------------|--|
| ABS(nr) | valoarea absolută a lui nr, rezultat întreg sau real; |
| ARCTAN(nr) | arctg(nr), nr întreg sau real, rezultat real; |
| COS(nr) | cos(nr), nr întreg sau real; |
| FRAC(nr) | partea fracționară a lui nr (întreg sau real), rezultat real; |
| INT(nr) | partea întreagă a lui nr (întreg sau real), rezultat întreg; |
| LN(nr) | logaritmul natural, nr întreg sau real, rezultat real; |
| SIN(nr) | sin(nr), nr întreg sau real, rezultat real; |
| SQR(nr) | pătratul lui nr, rezultat întreg sau real, funcție de nr; |
| SQRT(nr) | rădăcina pătrată din nr (întreg sau real), rezultat real; |
| PRED(nr) | predecesorul lui nr (orice tip scalar); |
| SUCC(nr) | succesorul lui nr (orice tip scalar); |
| ODD(nr) | valoare logică TRUE dacă nr (întreg) este impar, FALSE în caz contrar; |
| CHR(nr) | întoarce caracterul al cărui cod ASCII este nr (întreg); |

| | |
|---------------------|---|
| ORD(var) | întoarce numărul de ordine al valorii lui var în mulțimea definită de tipul lui var ; |
| ROUND(nr) | rotunjirea valorii variabilei nr (de tip real) la un întreg; |
| TRUNC(nr) | trunchierea valorii variabilei nr (real) la un întreg; |
| HI(i) | întoarce un întreg reprezentând valoarea octetului superior al valorii expresiei i ; |
| LO(i) | întoarce un întreg reprezentând valoarea octetului inferior al valorii expresiei i ; |
| KEYPRESSED | întoarce valoarea logică TRUE dacă o tastă a fost apasată, FALSE în caz contrar; |
| RANDOM | întoarce un număr aleator în intervalul [0,1], rezultat real; |
| RANDOM(nr) | întoarce un număr aleator aparținând mulțimii 0,1,2,...,nr, nr și rezultatul fiind întregi; |
| PARAMCOUNT | valoare întreagă reprezentând numărul de parametri transmiși programului prin linia de comandă; |
| PARAMSTR(i) | întoarce un STRING conținând al i -lea parametru din linia de comandă; |
| SIZE OF(num) | întoarce numărul de octeți ocupați de variabilă sau tipul de dată num , rezultat întreg; |
| SWAP(nr) | interschimbă octeții inferior și superior ai argumentului întreg nr , rezultat întreg; |
| UPCASE(car) | întoarce un caracter ce reprezintă echivalentul în litere mari al lui car (dacă există). |

Lista cuvintelor rezervate în TURBO PASCAL este prezentată în Anexa 3.5.2.

Pentru detalii în legătură cu mediul de programare TURBO PASCAL se poate consulta lucrarea [3].

3.2.3. Programe în limbajul TURBO PASCAL

EXEMPLUL 1 (Utilizarea recursiei):

Multe procese matematice iterative, mulți algoritmi de grafică computațională fac apel la recursie în locul iterațiilor. Limbajul TURBO PASCAL permite definirea unor funcții și proceduri recursive.

Fie funcția factorial:

$$f:N \rightarrow N, f(n) = \begin{cases} 1, & \text{dacă } n = 0 \\ n \cdot f(n-1), & \text{dacă } n \geq 1 \end{cases}$$

Valoarea acestei funcții într-un punct poate fi calculată cu ajutorul programului următor:

Programul FACT

```
Program FACT ;
type
    natural = 0..Maxint ;
var
    n,m : integer ;

function FR(n : natural) : natural ;
begin
    if n = 0 then FR := 1
    else FR := n * FR(n-1) ;
end ;

begin { programul principal }
    writeln(' Program pentru calculul factorialului ');
    writeln(' Precizati numarul n=' );
    readln(n) ;
    writeln(' Numarul n este :',n) ;
    m := FR(n) ; { Apelul functiei recursive }
    writeln(' Factorial de n este :',m) ;
end.
```

Același rezultat se obține înlocuind funcția FR care folosește recursia cu funcția FI care folosește iterația:

```
function FI (n : natural) ; natural ;
var
    i,ifi : natural ;
begin
```

```

    ifi := 1 ;
    if n > 1 then
        for i := 2 to n do ifi := i * ifi ;
    FI := ifi ;
end ;

```

EXEMPLUL 2 (Utilizarea variabilelor dinamice):

Programul care urmează realizează parcurgerea unui arbore binar în inordine, preordine și postordine. Nodurile sînt introduse cu ajutorul procedurii recursive ENTER. Programul folosește tipul de dată de tip referință, care permite generarea dinamică de variabile de tipul nod definite în program. Structura ptr introduce un tip de date care reprezintă o mulțime de variabile de tipul nod. Variabila rad este de tip referință.

Programul ARBORE

```

Program ARBORE ;
type
    ptr = ^nod ;
    nod = record
        info : char ; { numele nodului }
        legs : ptr ; { legatura la dreapta }
        legd : ptr ; { legatura la stinga }
    end ;
var
    rad : ptr ;
    ch : char ;
Procedure ENTER( var p:ptr) ;
{ Procedura de citire a arborelui }
begin
    read(ch) ;
    write(ch) ;
{ Pentru a indica sfirsitul introducerii nodurilor
  se tasteaza caracterul '.' }
    if ch <> '.' then begin
        { generarea unei variabile de tip nod }
        new(p) ;
        p^.info := ch ;
        enter(p^.legs) ;
        enter(p^.legd) ;
    end
end

```

```

                                else
                                    p := nil ;
                                end ;
Procedure INORDINE(p : ptr) ;
begin
    if p <> nil then begin
        inordine(p^.legs) ;
        write(p^.info) ;
        inordine(p^.legd)
    end;
end;
Procedure PREORDINE(p : ptr) ;
begin
    if p <> nil then begin
        write(p^.info) ;
        preordine(p^.legs) ;
        preordine(p^.legd) ;
    end ;
end ;
Procedure POSTORDINE(p : ptr) ;
begin
    if p <> nil then begin
        postordine(p^.legs) ;
        postordine(p^.legd) ;
        write(p^.info) ;
    end ;
end ;
begin { Programul principal }
    enter(rad) ;
    writeln ;
    preordine(rad) ;
    writeln ;
    inordine(rad) ;
    writeln ;
    postordine(rad) ;
end.

```

EXEMPLUL 3 (Utilizarea fişierelor):

Programul CRF creează un fişier care să conţină numele candidaţilor la un examen de admitere, notele obţinute şi media. Conţinutul fişierului este apoi listat.

 Programul CRF

```

Program CRF ;
  type student = record
      nrord : 1..200 ;
      nume   : string[20]
      note   : array[1..4] of real ;
      medie  : real ;
  end ;

  var
      s      : student ;
      disc   : file of student ;
      numei  : string[14] ;
      no     : array[1..4] of real ;
      j,l    : integer ;
      i      : boolean ;
      nu     : string[20] ;
  begin
      writeln(' Introduceti numele fisierului',
              ' de creat');
      readln(numei);
      assign(disc,numei);
      rewrite(disc);
      i := TRUE ;
      j := 0 ;
      while i do
          begin
              with s do begin
                  write(' Nume student:');
                  readln(nu); { Se tasteaza Enter pentru nume
                              atunci cind datele de intrare
                              s-au epuizat }
                  if nu <> "" then
                      begin
                          j := j + 1 ;
                          nrord := j ;
                          nume := nu ;
                          medie := 0 ;
                          for l:=1 to 4 do
                              begin
                                  write(' Nota ',l);

```

```
        readln(no[1]);
        medie:=medie+no[1];
    end;
    medie :=medie /4.0
    write(disc,s);
end
else
        i := FALSE ;
        end;
        end;
        close(disc); {Listeaza datele din fisier}
        reset (disc);
        while not eof(disc) do begin
            read(disc,s);
            write(s.num, s.medie:9:2);
        end;
        close(disc);
end.
```

3.3. Biblioteca grafică TURBO PASCAL

3.3.1. Prezentare generală

Mediul de programare TURBO PASCAL permite accesul la o bibliotecă de rutine grafice GRAPH care asigură o interfață eficientă între programator și dispozitivele grafice.

Biblioteca GRAPH este conținută în fișierul GRAPH.TPU care trebuie să se afle în directorul utilizator în momentul execuției programului de desen. Ea conține o mare varietate de rutine grafice, pornind de la cele orientate pe pixeli(PutPixel, PutImage, GetImage, GetPixel) pînă la cele de nivel înalt (SetWiewport, Circle, Bar3D, DrawPoly, Pieslice, etc).

Biblioteca de rutine grafice este independentă de dispozitiv.

Programatorul trebuie să declare în programul său că va utiliza biblioteca de rutine grafice GRAPH.TPU. Această declarație se realizează cu instrucțiunea:

```
USES GRAPH ;
```

Această instrucțiune realizează accesul programului utilizator la rutinele bibliotecii grafice.

Pentru a compila un program de desen sînt necesare programul sursă și fișierul TURBO.TPL.

Pentru a executa un program de desen este nevoie de driver-ul grafic (fișier de tip .BGI) corespunzător monitorului pe care se va vizualiza

desenul. În plus, dacă programul folosește mai multe forme de caractere (fonturi) sînt necesare unul sau mai multe fișiere de tip .CHR.

Dispozitivele grafice pentru care au fost realizate drivere reprezintă o mare parte din gama monitoarelor mono sau color care pot fi utilizate în configurația unui microcalculator compatibil IBM. Acest lucru asigură independența bibliotecii grafice față de dispozitivul de desen. Fiecare driver conține coduri, date și este stocat într-un fișier pe disc. Monitoarele pot lucra în două moduri: alfanumeric și grafic. Modul de lucru grafic poate avea la rîndul lui, mai multe moduri grafice. Un mod grafic este caracterizat de o anumită rezoluție și o anumită paletă de culori. În paragraful 3.3.8 este prezentată lista driver-elor și modurilor grafice folosite de biblioteca grafică GRAPH.

Rutinele bibliotecii GRAPH pot fi clasificate în funcție de efectul apelului asupra programului și/sau echipamentului, în următoarele categorii:

- de definire a sesiunii de desen;
- rutine de desenare texte;
- de trasare;
- rutine la nivel de pixel;
- rutine de determinare a contextului curent al desenului.

Transmiterea incorectă a parametrilor rutinelor grafice este semnalată de compilatorul TURBO PASCAL în timpul compilării programului sursă.

În anexa 3.5.3 este prezentată lista rutinelor și funcțiilor grafice din biblioteca grafică GRAPH.TPU.

În paragraful 3.3.9 este prezentat un program demonstrativ (compus din EX01, ..., EX11) în care sînt apelate rutine grafice din GRAPH.

3.3.2. Definirea sesiunii de desen

A defini o sesiune de desen înseamnă a preciza anumite elemente legate de dispozitivul pe care se va vizualiza obiectul de reprezentat (rezoluție, modul grafic) sau de desen (fixare viewport, fixare scală, inițializare punct curent, etc).

3.3.2.1. Inițializarea unei sesiuni de desen

InitGraph

Orice sesiune de desen trebuie să înceapă cu apelul rutinei InitGraph.

Această rutină identifică adaptorul grafic, încarcă și inițializează cel mai potrivit driver, trece sistemul în mod grafic și întoarce controlul

rutinelor grafice.

În momentul execuției acestei rutine grafice sînt definite mai multe tipuri de date și sînt definite o serie de constante. Tipurile de date definite, constantele rutinelor grafice sînt prezentate în paragraful 3.3.8.

Declarația acestei rutine grafice este:

```
Procedure InitGraph(var GraphDriver,  
                    GraphMode: integer; DriverPath: string );
```

GraphDriver reprezintă numărul driver-ului, iar GraphMode numărul modului grafic. DriverPath specifică directorul unde se găsesc fișierele de tip BGI (driver-ele grafice). Dacă DriverPath este nul, fișierele de tip BGI trebuie să fie în directorul curent.

Dacă GraphDriver=Detect se selectează automat modul grafic în funcție de tipul plăcii grafice.

RestoreCrtMode

După inițializare, dispozitivul grafic este în mod grafic. Cu ajutorul rutinei RestoreCrtMode se trece din modul de lucru grafic în mod alfanumeric (text). Declarația acestei rutine este:

```
Procedure RestoreCrtMode ;
```

SetGraphMode

Pentru a reveni în modul de lucru grafic se folosește rutina SetGraphMode, a cărei declarație este:

```
Procedure SetGraphMode( Mode : integer ) ;
```

Mode este numărul modului grafic în care se trece. Procedura șterge ecranul grafic și fixează atributele grafice la valoarea inițială (cele prezentate la procedura GraphDefaults).

Utilizatorul are posibilitatea gestionării driver-elor și modurilor grafice.

DetectGraph

Cu ajutorul rutinei DetectGraph se determină driver-ul grafic și modul grafic. Declarația acestei rutine este:

```
Procedure DetectGraph (var GraphDriver,  
                      GraphMode:integer) ;
```

GraphDriver și GraphMode sînt numerele driver-ului și modului grafic curente.

InstallUserDriver

Se poate instala un driver al utilizatorului folosind funcția InstallUserDriver:

```
Function InstallUserDriver (NameDriver: string;  
    AutoDetectPtr: pointer ): integer ;
```

unde NameDriver este numele fișierului care conține driver-ul, iar AutoDetectPtr reprezintă adresa funcției de detectare opțională.

GetModeRange

Valorile posibile pentru modul grafic corespunzătoare unui driver se pot afla cu rutina GetModeRange:

```
Procedure GetModeRange( GraphDriver: Integer;  
    var LoMode, HiMode: integer );
```

unde

- GraphDriver este numărul driver-ului cercetat;
- LoMode este modul grafic cu cea mai mică rezoluție;
- HiMode este modul grafic cu cea mai mare rezoluție.

GetDriverName

Numele driver-ului se obține folosind funcția GetDriverName:

```
Function GetDriverName : string ;
```

Funcția întoarce un șir conținând numele driver-ului curent.

GetGraphMode

Funcția GetGraphMode întoarce numărul modului grafic curent:

```
Function GetGraphMode : integer ;
```

Valoarea modului grafic variază între 0 și 5, depinzând de driver-ul grafic curent.

GetMaxMode

Funcția GetMaxMode furnizează numărul maxim de mod grafic ce poate fi trecut în SetGraphMode:

```
Function GetMaxMode : integer ;
```

GetModeName

Numele modului grafic se poate obține cu funcția GetModeName:

```
Function GetModeName (GraphMode:integer): string;
```

Exemple de utilizare a rutinelor InitGraph, RestoreCrtMode, SetGraphMode se găsesc în procedura INITIALIZARE (EX01) din programul demonstrativ prezentat în paragraful 3.3.9.

3.3.2.2. Detectarea erorilor

Codurile erorilor interne care apar la apelul rutinelor grafice sînt obținute cu ajutorul funcției `GraphResult`. Declarația acestei funcții este:

```
Function GraphResult : integer ;
```

Funcția `GraphResult` returnează codul de eroare al ultimei operații grafice executate. Codul erorii poate fi stocat într-o variabilă temporară. În funcție de codul erorii, prin program se poate hotărî continuarea sau oprirea execuției programului sau se descoperă cauza erorii. Codurile de eroare și semnificația lor sînt prezentate în paragraful 3.3.8.

Funcția `GraphErrorMsg` întoarce un mesaj de eroare pentru codul erorii obținut de `GraphResult`. Declarația funcției este:

```
Function GraphErrorMsg(CodEroare: integer): string;
```

În procedura `INITIALIZARE (EX01)` sînt apelate funcțiile `GraphResult` și `GraphErrorMsg`.

3.3.2.3. Coordonate

Desenul va fi realizat pe ecranul dispozitivului prin activarea pixelilor. Fiecare pixel se adresează prin coordonatele sale x și y la fel ca punctele din spațiul bidimensional. Colțul ecranului din stînga sus va avea coordonatele $(0,0)$. Valorile lui x (sau coloanele) cresc spre dreapta. Valoarea lui y (rîndurile) cresc în jos. Astfel, pentru dispozitivul grafic EGA (modul grafic, EGALo) coordonatele ecranului pentru cele patru colțuri și un punct din centrul ecranului sînt prezentate în figura următoare.

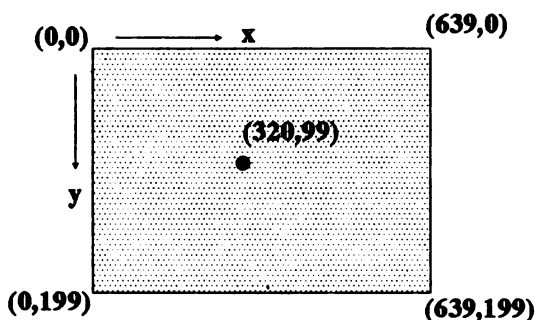


Figura 3-1.

Acesta este viewportul (vizorul) maxim pentru acest dispozitiv.

Notă: În toate rutinele care vor fi prezentate, coordonatele reprezintă coordonatele pixelilor și sînt numere întregi.

3.3.2.4. Fixare vizor(viewport). Decupare(clipping)

Imaginea poate fi realizată pe toată suprafața ecranului sau numai pe o porțiune dreptunghiulară a acestuia. Zona de pe ecran pe care se realizează desenul se fixează cu ajutorul procedurii SetViewport. Declarația acestei proceduri este:

```
Procedure SetViewport (x1,y1,x2,y2: word;
                      clip: boolean );
```

unde :

- (x1,y1) definește colțul din stînga sus al vizorului;
- (x2,y2) definește colțul din dreapta jos al vizorului;
- Dacă clip = TRUE atunci se realizează decuparea(clipping);
- Dacă clip = FALSE atunci nu se realizează decuparea.

Rutinele InitGraph, SetGraphMode, ClearDevice fixează vizorul la întreg ecranul grafic.

GetViewSettings

În cadrul unei sesiuni de desen se poate afla vizorul curent și dacă este activă decuparea cu ajutorul rutinei GetViewSettings, a cărei declarație este:

```
Procedure GetViewSettings(var ViewPort:ViewportType);
```

Tipul de date ViewPortType este definit în paragraful 3.3.8.

Cu ajutorul funcțiilor GetMaxX și GetMaxY se obțin numerele de pixeli pe linii și coloane care pot fi adresați cu driver-ul și modul grafic curente.

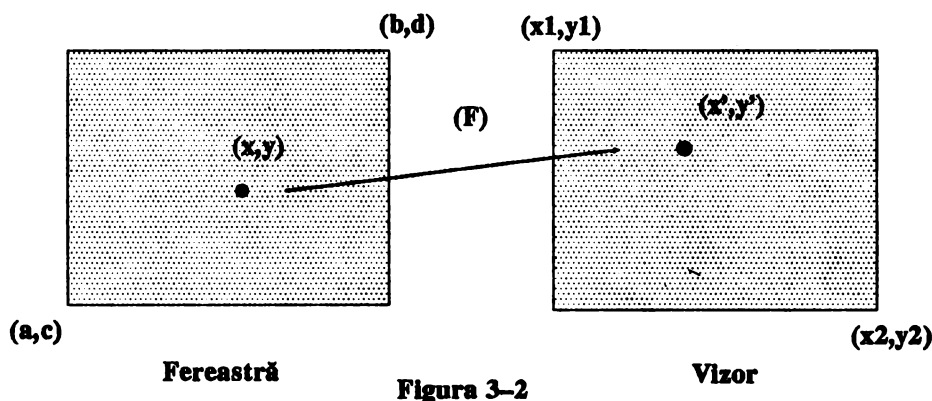
```
function GetMaxX : integer;
```

```
function GetMaxY : integer;
```

În procedura DESEN_BARA (EX10) apare apelul rutinei SetViewport.

Cum se face trecerea de la coordonatele utilizator la coordonatele ecran?

Fie $W = [a,b] \times [c,d] \subset \mathbb{R} \times \mathbb{R}$ spațiul utilizator al obiectului de desenat (fereastra) și $V = [x_1,x_2] \times [y_1,y_2]$ vizorul (suprafața pe care se desenează.)



Pentru a suprapune prin deformare imaginea din fereastră pe suprafața de desen (vizor) este nevoie să aplicăm următoarea transformare:

$$\begin{aligned} x' &= \left[\frac{(x-a)(x_2-x_1)}{(b-a)} + \frac{1}{2} \right] \\ y' &= y_2 - \left[\frac{(y-b)(y_2-y_1)}{(d-c)} + \frac{1}{2} \right] \end{aligned} \quad (F)$$

unde:

- (x, y) sînt coordonatele carteziene ale punctului de pe obiectul de desenat;
- (x', y') sînt coordonatele ecran ale punctului care se desenează;
- $[v]$ este partea întreagă a numărului v .

Această transformare nu este injectivă, mai multe puncte din spațiul utilizator putînd fi reprezentate prin același pixel.

Înainte de apelul rutinelor grafice, tuturor coordonatelor punctelor din spațiul utilizator li se aplică transformarea (F).

Toate coordonatele sînt relative la fereastra curentă (mai puțin cele din `setViewPort`).

3.3.2.5. Punct curent

Multe sisteme grafice suportă noțiunea de punct curent. Punctul curent grafic este similar cu conceptul de cursor în modul de lucru text, cu diferența că punctul curent grafic nu este vizibil. Punctului curent nu i se aplică decuparea. Coordonatele punctului curent sînt în interiorul vizorului maxim.

GetX și GetY

Pentru a afla coordonatele punctului curent se folosesc funcțiile GetX și GetY:

```
function GetX : integer;
function GetY : integer;
```

În procedura TEXT din programul demonstrativ apar apelurile funcțiilor GetX și GetY.

Punctul curent se modifică după apelul rutinelor de desen.

Rutinele InitGraph, SetGraphMode, ClearDevice și setViewport fixează punctul curent în (0,0).

MoveTo

Programatorul poate schimba punctul curent cu ajutorul rutinei MoveTo a cărei declarație este:

```
Procedure MoveTo( x1, y1 : integer ) ;
```

Rutina mută punctul curent în (x1,y1) fără a trasa ceva. Coordonatele x1, y1 sînt considerate în vizorul (Viewport-ul) activ.

MoveRel

Modificarea punctului curent se poate indica și în coordonate relative:

```
Procedure MoveRel( Dx, Dy : integer ) ;
```

Punctul curent este traslatat cu vectorul (Dx,Dy) față de vechiul punct curent. Mutarea cursorului se face fără desenare.

Toate rutinele grafice care nu specifică punctul unde începe desenarea vor folosi punctul curent drept început al desenului.

3.3.2.6. Scalarea și rotația

În unele aplicații e nevoie ca anumite obiecte să fie rotite, mărite sau micșorate. Aceste operații pot fi aplicate atît spațiului real cît și spațiului desenului.

Pentru rotația punctului (x,y) în jurul punctului (x_r,y_r) cu unghiul „a” se folosesc relațiile:

$$\begin{aligned}x' &= (x - x_r) \cdot \cos a - (y - y_r) \cdot \sin a + x_r \\ y' &= (x - x_r) \cdot \sin a + (y - y_r) \cdot \cos a + y_r\end{aligned}$$

unde (x',y') sînt coordonatele punctului după rotație.

Scalarea în raport cu originea se realizează prin înmulțirea coordonatelor cu factorul de scală. Dacă factorul de scală este mai mare decît 1 obiectul se mărește, dacă factorul de scală este mai mic decît 1 obiectul se micșorează.

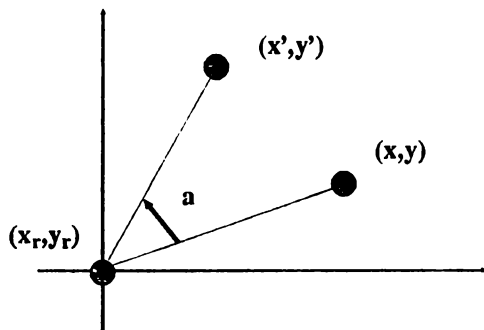


Figura 3-3

GetAspectRatio

Pentru a păstra forma obiectelor (pătrate, cercuri) este necesară o anumită scalare. Există o rutină care indică factorul de scalare în așa fel ca pătratele, cercurile să-și păstreze forma după desenare. Se folosește rutina `GetAspectRatio`:

```
Procedure GetAspectRatio (var Xasp, Yasp: word );
```

Coordonata x rămîne nemodificată, iar coordonata y se înmulțește cu $Yasp/Xasp$.

SetAspectRatio

Utilizatorul poate să-și fixeze factorii de scalare pe x și y folosind rutina `SetAspectRatio`:

```
Procedure SetAspectRatio ( Xasp, Yasp : word ) ;
```

În procedura `Diagrama_circulara (EX07)` apare un exemplu de utilizare a rutinei `GetAspectRatio`.

3.3.2.7. Fixarea culorii

Există rutine pentru fixarea culorii fondului și rutine pentru fixarea culorii desenului.

GetMaxColor

Numărul culorilor disponibile se obține cu ajutorul funcției:

```
Function GetMaxColor : word ;
```

SetBkColor

Pentru fixarea culorii fondului se folosește rutina `SetBkColor`, a cărei declarație este:

```
Procedure SetBkColor(Color : word) ;
```

unde `color` ia valori între 0 și `GetMaxColor`.

GetBkColor și GetColor

Culorile curente pentru fond și desen se obțin folosind următoarele funcții:

```
Function GetBkColor : word ;
Function GetColor : word ;
```

GetPalette și GetPaletteSize

Paleta de culori disponibile se poate afla apelând rutina GetPalette:

```
Procedure GetPalette(var Palette:PaletteType);
```

unde PaletteType este definit în paragraful 3.3.8.

Paleta de culori diferă în funcție de driver și modul grafic curent.

Dimensiunea paletei de culori se află cu rutina:

```
Function GetPaletteSize : integer ;
```

SetPalette și SetAllPalette

Paleta de culori poate fi modificată cu ajutorul rutinelor SetPalette și SetAllPalette.

SetPalette are declarația :

```
Procedure SetPalette(ColorNum: word;
                    Color: shortint);
```

Această rutină schimbă culoarea ColorNum cu culoarea Color.

SetAllPalette modifică mai multe culori și are declarația:

```
Procedure SetAllPalette(var Palette:PaletteType);
```

unde PaletteType este definit în 3.3.8.

GetDefaultPalette

Pentru a reveni la paleta definită la apelul rutinei InitGraph se folosește rutina GetDefaultPalette:

```
Procedure GetDefaultPalette(var Palette:PaletteType);
```

unde tipul PaletteType este definit în paragraful 3.8.

SetColor și SetRGBPalette

Culoarea cu care se realizează desenul se fixează cu ajutorul rutinei SetColor:

```
Procedure SetColor(Color : word) ;
```

unde Color reprezintă numărul culorii în paleta driver-ului dispozitivului și a modului grafic curent.

Rutina SetRGBPalette fixează intensitățile culorilor de bază:

```
SetRGBPalette (ColorNum,Red,Green,Blue: integer);
```

unde:

- `ColorNum` precizează numărul culorii ce se definește:
- `Red`, `Green`, `Blue` sînt intensitățile culorilor roșu, verde și albastru (valori cuprinse între 0 și 63).

3.3.2.8. Închiderea sesiunii de desen

Sfîrșitul sesiunii de desen se anunță cu rutina `CloseGraph` care trece dispozitivul în modul de lucru alfanumeric.

Declarația rutinei este:

`Procedure CloseGraph ;`

Toate rutinele grafice pot fi apelate între apelul lui `InitGraph` și cel al rutinei `CloseGraph`.

3.3.3. Desenarea textelor

Sistemul oferă posibilitatea trasării unui text în două moduri:

- reprezentarea caracterului printr-o matrice de 8×8 pixeli;
- reprezentarea prin caractere trasate, definite printr-o serie de vectori care indică sistemului grafic cum să deseneze caracterul.

Avantajul folosirii caracterelor trasate este vizibil cînd se trasează caractere largi. Deoarece caracterul este definit prin vectori, acesta va apărea clar și de calitate cînd semnul este mare. Reprezentarea prin matrice de 8×8 pixeli este indicată pentru caractere de dimensiune mică.

3.3.3.1. Setări pentru trasarea caracterelor

SetTextJustify

Poziția textului este controlată cu ajutorul rutinei `SetTextJustify`:

`Procedure SetTextJustify(Horiz,Vert: word);`

Valorile pe care le pot lua `Horiz` și `Vert` sînt prezentate în paragraful 3.3.8.

SetTextStyle

Scalarea și selecția formei caracterelor este realizată cu rutina `SetTextStyle`:

`Procedure SetTextStyle(Font: word;`

`Direction: word; Size: word);`

Parametrul `Font` indică tipul de caractere folosit la trasare.

Valorile pe care le pot lua parametrii din `SetTextStyle` sînt prezentate în paragraful 3.3.8.

SetUserCharSize

Utilizatorul își poate fixa dimensiunile caracterelor folosind rutina `SetUserCharSize` a cărei declarație este:

```
Procedure SetUserCharSize(MultX,DivX,MultY,DivY);
```

unde:

- raportul `MultX:DivX` se va înmulți cu lățimea normală;
- raportul `MultY:DivY` se va înmulți cu înălțimea normală.

Dacă dimensiunile caracterelor sînt fixate de utilizator atunci în rutina `SetTextStyle`, mărimea caracterelor se indică prin variabila `UserCharSize` definită în paragraful 3.3.8.

GetTextSettings

Pentru a cunoaște caracteristicile cu care se trasează un text se pot folosi rutinele: `GetTextSettings` (pentru formă, direcție, mărime, aliniere), `TextWindth` (pentru lățime), `TextHeight` (pentru înălțime).

```
Procedure GetTextSettings
```

```
(var TextInfo:TextSettingsType);
```

```
Function TextWindth(var text : string) : word ;
```

```
Function TextHeight(var text : string) : word ;
```

Tipul de date `TextSettingsType` este definit în paragraful 3.3.8.

În procedura `TEXT (EX03)` sînt apelate rutinele grafice descrise mai sus.

3.3.3.2. Trasarea unui text

OutText și OutTextXY

Pentru trasarea unui text se folosesc rutinele `OutText` și `OutTextXY`.

`OutText` trasează un text în punctul curent. Declarația rutinei este:

```
Procedure OutText(Text : string) ;
```

Punctul curent se modifică corespunzător, cu înălțimea textului pe `y` și cu lungimea textului pe `x`.

`OutTextXY` trasează un text începînd din punctul de coordonate `(x,y)`. Declarația rutinei este:

```
Procedure OutTextXY (x,y:integer; Text:string);
```

Exemple de utilizare în `EX02`, `EX03`, `EX04`, etc.

3.3.3.3. Trasări numerice

Pentru editările numerice propunem soluția care urmează.

Pentru a edita un număr întreg acesta se transformă mai întîi, într-un șir de caractere cu ajutorul funcției `Int2Str` definită astfel:

```
function Int2Str(L : longint ) : string ;  
  var s : string ;  
  begin  
    Str(L,s) ;  
    Int2Str := s ;  
  end ;
```

Funcția transformă numărul întreg L într-un șir s care poate fi trasat cu OutText sau OutTextXY.

Pentru trasarea unui numar real val cu nrz zecimale se poate folosi următoarea procedură:

```
Procedure EditReal(var val:real; var nrz:integer);  
var  
  valintreg, valzec : integer ;  
begin  
  valintreg:=INT(val); { Partea intreaga }  
  If nrz := 0 then  
    OutText(Int2Str(valintreg))  
  else  
    begin  
      { ... Calculeaza primele "nrz" cifre  
        ... ale partii zecimale }  
      valzec := round(val*exp(nrz*LN(10.0))-valintreg*  
        Exp(nrz*LN((10.0))) ;  
      OutText(Int2Str(valintreg)+'.'+Int2Str(valzec));  
    end;  
end;
```

Pentru a fixa locul de trasare a numărului se folosește rutina MoveTo.

Deoarece nu există funcția a^x s-a folosit formula:

$$a^x = e^{x \ln(a)}$$

Funcția Int2Str este definită în programul demonstrativ și este apelată în procedura TipLinie (EX05).

3.3.4. Trasări

3.3.4.1. Trasări de segmente

Line

Pentru trasarea unui segment se folosește rutina Line. Declararea acestei rutine este:

```
Procedure Line(x1, y1, x2, y2 : integer) ;
```

Rutina Line trasează un segment de la punctul (x1,y1) la punctul (x2,y2).

LineTo

Rutina `LineTo` trasează un segment de la punctul curent pînă la punctul (x1,y1). Declaraarea rutinei este:

```
Procedure LineTo(x1, y1 : integer) ;
```

LineRel

Rutina `LineRel` trasează un segment de la punctul curent pînă la punctul situat la distanța relativă (Dx,Dy) față de punctul curent. Are următoarea declarație:

```
Procedure LineRel( Dx, Dy : integer ) ;
```

Tipul liniei și culoarea sînt definite cu ajutorul rutinelor `SetLineStyle` și `SetColor`.

SetLineStyle

Cu ajutorul rutinei `SetLineStyle` se definesc stilul, forma și grosimea liniei:

```
Procedure SetLineStyle(LineStyle: word;  
Pattern: word; Thickness: word);
```

Valorile pentru `LineStyle` și `Thickness` sînt prezentate în paragraful 3.3.8. Pentru `LineStyle=4` (`UserBitLn`) valoarea lui `Pattern` va fi un număr hexazecimal pe 16 biți. Pentru celelalte valori ale lui `LineStyle`, `Pattern` este 0.

Rutina are efect și asupra rutinelor `LineTo`, `Rectangle`, `DrawPoly`, `Arc`, `Circle`, `Ellipse`. Pentru completarea parametrilor se consultă paragraful 3.3.8.

SetWriteMode

Există procedura `SetWriteMode` care indică modul de desenare al liniei:

```
Procedure SetWriteMode(WriteMode: integer);
```

unde `WriteMode` poate lua una din valorile:

- `XorPut = 0`
- `XorPut = 1`

Pentru `WriteMode = 0` linia se trasează peste ecran, apare în întregime. Pentru `WriteMode = 1` pixelii aparținînd liniei care erau deja activați vor fi dezactivați (linia apare cu întreruperi).

GetLineSettings

Pentru a afla caracteristicile liniilor care se trasează se utilizează rutina `GetLineSettings`:

`GetLineSettings(var LineInfo:LineSettingsType)`
unde `LineSettingsType` reprezintă un tip de date descris în paragraful 3.3.8.

3.3.4.2. Trasări de dreptunghiuri, poligoane, cercuri, arce de cerc și ellipse

Rectangle

Pentru trasarea unui dreptunghi se folosește rutina `Rectangle`:
`Procedure Rectangle(x1,y1,x2,y2:integer);`
unde `(x1,y1)` este colțul din stînga sus, iar `(x2,y2)` este colțul din dreapta jos.

DrawPoly

O polilinie se trasează cu ajutorul rutinei `DrawPoly` declarată astfel:

`Procedure DrawPoly(NumPoints: word;`
`var PolyPoints);`

unde:

- `NumPoints` reprezintă numărul de puncte din `PolyPoints`;
- `PolyPoints` conține coordonatele vîrfurilor liniei poligonale. `PolyPoints` este variabilă fără tip (untyped).

Pentru a desena o linie poligonală închisă trebuie să avem:

`PolyPoints[n] = PolyPoints[1]`

Circle

Rutina `Circle` desenează un cerc:

`Procedure Circle(x,y: integer; Radius: word);`

Rutina trasează un cerc cu centrul în `(x,y)` și raza `Radius`.

Arc

Un arc de cerc se trasează cu rutina `Arc`:

`Procedure Arc(x,y: integer;`
`StAngle,EndAngle,Radius: word);`

Rutina trasează arcul de cerc din cercul de centru `(x,y)` și raza `Radius` cuprins între unghiurile `StAngle` și `EndAngle`.

GetArcCoords

Există o rutină care furnizează coordonatele capătului ultimului arc de cerc trasat:

`Procedure GetArcCoords(var ArcCoords:ArcCoordsType);`
unde `ArcCoordsType` este definit în 3.3.8.

Ellipse

Rutina **Ellipse** trasează un arc de elipsă:

```
Procedure Ellipse (x,y: integer ;
                  stAngle,EndAngle,xRadius,yRadius: word);
```

Rutina trasează un arc de elipsă cu centrul în (x,y) avînd axele **xRadius**, **yRadius**, cuprins între unghiurile **stAngle** și **EndAngle**. Dacă **stAngle** = 0 și **EndAngle** = 359 se trasează întreaga elipsă. Unghiurile se dau în grade hexagesimale, în sens invers acelor de ceasornic.

3.3.4.3. Hașuri

SetFillStyle și SetFillPattern

Utilizatorul are posibilitatea hașurării unui domeniu. Tipul de hașură și culoarea se stabilesc cu ajutorul rutinelor **SetFillStyle** și **SetFillPattern**:

```
Procedure SetFillStyle (Pattern:word; Color:word);
Procedure SetFillPattern (Pattern:FillPatternType;
                          Color:word);
```

Pattern este tipul modelului de hașurare. **FillPatternType** este definit în paragraful 3.3.8. Aceste rutine au efect asupra rutinelor **FillPoly**, **FloodFill**, **Bar**, **Bar3D**, **PieSlice**.

FillPoly

Rutina **FillPoly** desenează și umple un poligon. Are următoarea declarație:

```
Procedure FillPoly(NumPoints : word ; var PolyPoints);
unde:
```

- **NumPoints** reprezintă numărul de coordonate în **PolyPoints**;
- **PolyPoints** conține coordonatele vîrfurilor poligonului; **PolyPoints** este o variabilă fără tip.

Rutina **FillPoly** calculează intersecțiile orizontale și apoi umple poligonul folosind stilul și culoarea de hașurare definite de **SetFillStyle** sau **SetFillPattern**.

FloodFill

Rutina **FloodFill** hașurează o arie mărginită cu modelul de hașură curent. Declarația sa este următoarea:

```
Procedure FloodFill( x, y, Border : word ) ;
```

unde:

- (x,y) este un punct din aria respectivă;
- **Border** este culoarea zonei de hașurat.

FillEllipse

Rutina FillEllipse desenează și umple o elipsă cu centrul în (x,y) avînd razele XRadius și YRadius:

```
Procedure FillEllipse(x,y: integer;
    XRadius,YRadius: word);
```

Rutina Sector desenează și umple un sector de elipsă:

```
Procedure Sector(x,y: integer;
    StAngle,EndAngle,XRadius,YRadius: word);
```

unde:

- (x,y) este centrul elipsei;
- StAngle, EndAngle sînt unghiurile de început și sfîrșit;
- XRadius și YRadius sînt razele pe x și pe y.

Bar

Rutina Bar desenează un dreptunghi hașurat cu modelul curent și culoarea curentă. Declarația rutinei este:

```
Procedure Bar( x1, y1, x2, y2 : integer );
```

Rutina umple dreptunghiul cu colțul stînga-sus în (x1,y1) și colțul dreapta-jos în (x2,y2). Rutina este apelată în procedura Diagrama_dreptunghiulara (EX09).

Bar3D

Rutina Bar3D desenează o bară tridimensională (paralelipiped dreptunghic) cu modelul de hașură și culoarea curente. Declarația sa este:

```
Procedure Bar3D(x1,y1,x2,y2: integer;
    Depth: word; Top: boolean) ;
```

unde:

- (x1, y1) este colțul stînga-sus din față al corpului;
- (x2, y2) este colțul dreapta-jos din față al corpului;
- Depth este adîncimea barei;
- Top = TRUE, bara este acoperită;
- Top = FALSE, bara este neacoperită;

PieSlice

Rutina PieSlice permite desenarea unei diagrame circulare. Declarația ei este:

```
Procedure PieSlice(x, y : integer ;
    StAngle, EndAngle, Radius : word);
```

Rutina desenează și hașurează o porțiune din diagrama circulară cu centrul în (x,y), de rază Radius, cuprinsă între StAngle și EndAngle.

Culoarea cercului este culoarea curentă. Modelul și culoarea hașurării sînt date de `setFillstyle` sau de `setFillPattern`. Un exemplu de utilizare se poate vedea în procedura `Diagrama_circulara (EX07)`.

3.3.5. Rutine la nivel de pixel

Aceste rutine sînt foarte utile în realizarea unor jocuri pe calculator și în special în animația pe calculator.

PutPixel

Rutina `PutPixel` trasează un punct (activează un pixel de pe ecran). Declarația ei este:

```
Procedure PutPixel( x,y: integer; Pixel: word);
```

Rutina desenează un punct de culoarea `Pixel` în punctul `(x,y)`.

GetPixel

Funcția `GetPixel` returnează culoarea pixelului de coordonate `(x,y)`. Declarația ei este:

```
Function GetPixel( x, y : integer ) : word ;
```

GetImage

Pentru a salva porțiuni dreptunghiulare de pe desen se folosește rutina `GetImage`. Declarația ei este:

```
Procedure GetImage(x1,y1,x2,y2: word; var BitMap);
```

unde:

- `x1, y1, x2, y2` definesc regiunea dreptunghiulară ce se salvează;
- `BitMap` este o variabilă de tip pointer care indică adresa la care se salvează imaginea formată din biți.

Dimensiunea zonei care se stochează nu trebuie să depășească 64 Ko.

ImageSize

Funcția `ImageSize` calculează numărul de pixeli necesari funcției `GetImage` pentru a salva regiunea dreptunghiulară definită de `(x1,y1)` și `(x2,y2)`:

```
Function ImageSize (x1, y1, x2, y2 : word) : word ;
```

Numărul de biți este dat de expresia: $(x2-x1+1)*(y2-y1+1)$

PutImage

Imaginea salvată cu `GetImage` poate fi redesenată într-o altă poziție cu ajutorul rutinei `PutImage`:

```

Procedure PutImage(x,y: integer;
    var BitMap,BitBlt: word);

```

unde:

- (x,y) este colțul din stînga-sus al regiunii dreptunghiulare unde se desenează;
- BitMap este adresa unde se găsește memorată imaginea;
- BitBlt indică un operator binar între punctele regiunii de suprapus și punctele ecranului, ale cărui valori sînt definite în paragraful 3.3.8. În procedura **Animatie_pe_calculator** (EX08) sînt apelate aceste rutine.

3.3.6. Diverse

Utilizatorul poate seta sistemul grafic la valorile implicite (de la inițializare) folosind rutina **GraphDefaults**:

```

Procedure GraphDefaults;

```

Valorile implicite ale sistemului grafic la inițializare sînt:

- atributele ferestrei grafice:


```

                (x1, y1) = (0, 0);
                (x2, y2) = (GetMaxX, GetMaxY);
                Clip     = True;
            
```
- atributele liniei:


```

                LineStyle = SolidLn;
                Pattern   = $FF;
                Thickness = NormWidth;
            
```
- atributele hașurării:


```

                Pattern = SolidFill;
                Color   = GetMaxColor;
            
```
- atributele textului:


```

                Font      = DefaultFont;
                Direction = HorizDir;
                CharSize  = 1;
                Horiz     = LeftText;
                Vert      = BottomText;
            
```
- atributele paletei:


```

                Size      = GetMaxColor+1;
                Colors[i] = i; { i=0, 1, ..., Size-1 };
            
```
- culoarea fondului = Colors[0];
- culoarea textului = Colors[GetMaxColor].

Pentru umplerea unui poligon sistemul folosește un buffer a cărui mărime poate fi modificată cu rutina **SetGraphBufferSize**:

```

Procedure SetGraphBufSize( BufSize : word ) ;

```

Valoarea implicită a bufferului este de 4 Ko, suficient de mare pentru a umple un poligon cu 655 vîrfuri.

Sistemul oferă posibilitatea trimiterii ieşirilor grafice către o pagină din memorie fără ca acestea să mai apară pe ecran, conţinutul acelei pagini putînd fi apoi vizualizat pe ecran. Această facilitate este folosită des în animaţia pe calculator.

Sînt disponibile rutinele `SetActivePage` şi `SetVisualPage`. Rutina `SetActivePage` defineşte pagina către care sînt îndreptate ieşirile grafice:

```
Procedure SetActivePage( Page : word );
```

Toate ieşirile grafice care urmează după această instrucţiune vor fi stocate în pagina cu numărul `Page`. Vizualizarea imaginii din pagina `Page` se face cu rutina `SetVisualPage`:

```
Procedure SetVisualPage( Page : word ) ;
```

Procedura permite vizualizarea alternativă a paginilor, lucru des folosit în animaţia pe calculator.

Folosirea mai multor pagini este acceptată numai cu plăci grafice EGA (256 Ko), VGA şi Hercules.

3.3.7. Ştergerea şi copierea ecranului

ClearDevice

Pentru ştergerea întregului ecran se foloseşte rutina `ClearDevice`:

```
Procedure ClearDevice ;
```

Rutina fixează punctul curent, paleta, culoarea, viewport-ul la valorile implicite.

ClearViewPort

Se poate şterge numai o porţiune dreptunghiulară de pe ecran. Cu ajutorul rutinei `SetViewPort` se fixează porţiunea care se va şterge, iar pentru ştergerea efectivă se foloseşte rutina `ClearViewPort` a cărei declaraţie este:

```
Procedure ClearViewPort ;
```

Delay

Pentru a păstra o imagine (desen) pe ecran se foloseşte instrucţiunea `DELAY` din biblioteca CRT (CRT trebuie să apară în instrucţiunea `USES`):

```
Procedure DELAY(ms : word) ;
```

unde `ms` reprezintă numărul de milisecunde de aşteptare. Se pot folosi de asemenea instrucţiunile:

```
REPEAT UNTIL KEYPRESSED ; ch := ReadKey ;
```

unde `ch` trebuie declarată variabilă de tip `char`. Imaginea se păstrează pe

ecran pînă cînd se va apăsa pe o tastă oarecare.

Pentru copierea unui desen de pe ecran pe imprimanta grafică (de exemplu RCD 9535) se procedează astfel:

- se execută comanda GRAPHICS din MS-DOS;
- se execută programul de desen al utilizatorului;
- după ce apare desenul pe ecran se apasă simultan tastele Shift și PrintScreen.

3.3.8. Constante, tipuri de date și variabile definite în biblioteca grafică GRAPH.TPU

```

{*****}
{
{      Turbo Pascal Version 5.5      }
{      Graph Interface Unit          }
{                                     }
{*****}

unit Graph ;
interface
CONST
  {Declaratii de constante}
  {*****}
  { Codurile de eroare din GraphResult: }
  grOk                = 0; { nu exista eroare }
  grNoInitGraph       = -1; { nu exista fisier BGI }
  grNotDetected       = -2; { nu gaseste hardware grafic }
  grFileNotFound      = -3; { lipseste driver-ul }
  grInvalidDriver     = -4; { driver invalid }
  grNoLoadMem         = -5; { memorie insuficienta pentru
                             driver }
  grNoScanMem         = -6; { memorie insuficienta }
  grNoFloodMem        = -7; { memorie insuficienta }
  grFontNotFound      = -8; { fisier de tip CHR negasit }
  grNoFontMem         = -9; { memorie insuficienta pentru
                             setul de caractere }
  grInvalidMode       = -10; { mod grafic necorespunzator
                             pentru driver-ul ales }
  grError             = -11; { eroare grafica }
  grIOError           = -12; { eroare intrare/iesire
                             grafica }
  grInvalidFont       = -13; { fisier cu setul de
                             caractere necorespunzator }
  grInvalidFontNum    = -14; { numar set de caractere
                             necorespunzator }

  { Definire drivere grafice }
  CurrentDriver = -128; { pentru rutina
  GetModeRange }
  Detect        = 0; { cere modul grafic cu cea
                     mai mare rezolutie, corespunzator plachetei grafice }
  CGA           = 1;

```

```

MCGA      = 2;
EGA       = 3;
EGA64     = 4;
EGAMono   = 5;
IBM8514   = 6;
HercMono  = 7;
ATT400    = 8;
VGA       = 9;
PC3270    = 10;

```

```
{ Moduri grafice pentru fiecare driver }
```

| { Mod grafic } | { Rezolutie } | { Numar pagini} | { Numar culori} |
|----------------|---------------|-----------------|-----------------|
| CGAC0 | = 0; 320x200 | 1 | 16 |
| CGAC1 | = 1; 320x200 | 1 | 16 |
| CGAC2 | = 2; 320x200 | 1 | 16 |
| CGAC3 | = 3; 320x200 | 1 | 16 |
| CGAHi | = 4; 640x200 | 1 | 2 |
| MCGAC0 | = 0; 320x200 | 1 | 16 |
| MCGAC1 | = 1; 320x200 | 1 | 16 |
| MCGAC2 | = 2; 320x200 | 1 | 16 |
| MCGAC3 | = 3; 320x200 | 1 | 16 |
| MCGAMed | = 4; 640x200 | 1 | 2 |
| MCGAHi | = 5; 640x480 | 1 | 2 |
| EGALo | = 0; 640x200 | 4 | 16 |
| EGAHi | = 1; 640x350 | 2 | 16 |
| EGA64Lo | = 0; 640x200 | 1 | 16 |
| EGA64Hi | = 1; 640x350 | 1 | 4 |
| EGAMonoHi | = 3; 640x350 | 1/2 | 2 |
| HercMonoHi | = 0; 720x348 | 2 | 2 |
| ATT400C0 | = 0; 320x200 | 1 | 16 |
| ATT400C1 | = 1; 320x200 | 1 | 16 |
| ATT400C2 | = 2; 320x200 | 1 | 16 |
| ATT400C3 | = 3; 320x200 | 1 | 16 |
| ATT400Med | = 4; 640x200 | 1 | 2 |
| ATT400Hi | = 5; 640x400 | 1 | 2 |
| VGALo | = 0; 640x200 | 4 | 16 |
| VGAMed | = 1; 640x350 | 2 | 16 |
| VGAHi | = 2; 640x480 | 1 | 16 |
| PC3270Hi | = 0; 720x350 | 1 | 2 |
| IBM8514Lo | = 0; 640x480 | | 256 |
| IBM8514Hi | = 1; 1024x768 | | 256 |

```
{ Culori pentru SetPalette/SetAllPalette: }
```

```

Black     = 0;
Blue      = 1;
Green     = 2;
Cyan      = 3;
Red       = 4;
Magenta   = 5;
Brown     = 6;
LightGray = 7;
DarkGray  = 8;
LightBlue = 9;
LightGreen = 10;

```

```

        LightCyan    = 11;
        LightRed     = 12;
        LightMagenta = 13;
        Yellow       = 14;
        White        = 15;

{ Culori EGA }
    EGABlack      = 0; { culori inchise }
    EGABlue       = 1;
    EGAGreen      = 2;
    EGACyan       = 3;
    EGARed        = 4;
    EGAMagenta    = 5;
    EGABrown      = 20;
    EGALightgray  = 7;
    EGADarkgray   = 56; { culori deschise }
    EGALightblue  = 57;
    EGALightgreen = 58;
    EGALightcyan  = 59;
    EGALightred   = 60;
    EGALightmagenta = 61;
    EGAYellow     = 62;
    EGAWhite      = 63;

{ Stiluri de linie si grosimi pentru Get/SetLineStyle }
{ Parametrul "LineStyle": }
    SolidLn      = 0; {-----}
    DottedLn     = 1; {- . - . - . -}
    CenterLn     = 2; {_. ._. ._. ._.}
    DashedLn     = 3; {__ __ __ __}
    UserBitLn    = 4; {Stil de linie definit de utilizator}
{ Parametrul "Thickness": }
    NormWidth    = 1;
    ThickWidth   = 3;

{ Tipuri de caractere pentru Set/GetTextStyle }
    DefaultFont  = 0; { caractere pe 8x8 biti }
    TriplexFont  = 1; { caractere marite de 3 ori }
    SmallFont    = 2; { caractere micșorate }
    SansSerifFont = 3;
    GothicFont   = 4; { caractere gotice }

{ Pozitie text -- parametrul "Direction" }
    HorizDir     = 0; { orizontal }
    VertDir      = 1; { vertical }
{ Dimensiune caractere utilizator }
    UserCharSize = 0;
{ Constante pentru decupare: }
    ClipOn       = true; { functioneaza decuparea }
    ClipOff      = false; { nu functioneaza decuparea }
{ Constante pentru rutina Bar3D: }
    TopOn        = true;
    TopOff       = false;
{ Stiluri de hasurare pentru Get/SetFillStyle: }

```



```

EmptyFill      = 0; { coloreaza cu culoarea de fond }
SolidFill      = 1; { nuanta uniforma temperata }
LineFill       = 2; { --- linii orizontale }
LtSlashFill    = 3; { model /// }
SlashFill      = 4; { model /// gros }
BkSlashFill    = 5; { model \\\ gros }
LtBkSlashFill  = 6; { model \\\ }
HatchFill      = 7; { model patratele }
XHatchFill     = 8; { model patratele oblice }
InterleaveFill = 9; { hasurare cu puncte dese }
WideDotFill    = 10; { hasurare cu puncte rare }
CloseDotFill   = 11; { punctat intermediar }
UserFill       = 12; { model definit de utilizator }

{ Operatorul BitBlt pentru PutImage: }
NormalPut      = 0; { MOV }
CopyPut        = 0; { MOV }
XORPut         = 1; { XOR }
OrPut          = 2; { OR }
AndPut         = 3; { AND }
NotPut         = 4; { NOT }

{ Reglajul orizontal si vertical pentru SetTextJustify: }
CenterText     = 1; { centru }
LeftText       = 0; { stinga }
RightText      = 2; { dreapta }
BottomText     = 0; { jos }
TopText        = 2; { sus }

{ Cod culoare maxima }
MaxColors      = 15;

TYPE
  { Definire tipuri de date }
  { Atributele paletii de culori }
  PaletteType = record
    Size : Byte;
    Colors : array[0..MaxColors] of Shortint;
  end;

  { Atributele liniei, obtinute cu GetLineSettings }
  LineSettingsType = record
    LineStyle : Word;
    Pattern : Word;
    Thickness : Word;
  end;

  { Atributele textului, obtinute cu GetTextSettings }
  TextSettingsType = record
    Font : Word;
    Direction : Word;
    CharSize : Word;
    Horiz : Word;
    Vert : Word;
  end;

  { Atributele hasurarii }
  FillSettingsType = record
    Pattern : Word;
    Color : Word;
  end;

```

```

    { Model hasurare (SetFillPattern) }
        FillPatternType = array [1..8]
        of Byte;
    { Coordonata punctului }
        PointType = record
            X, Y : integer;
        end;
    { Atributele ferestrei grafice }
        ViewPortType = record
            x1, y1 : integer; { coltul stinga`sus }
            x2, y2 : integer; { coltul dreapta jos }
            Clip : Boolean; { decuparea }
        end;
    { Atributele arcului }
        ArcCoordsType = record
            X, Y : integer ; { centrul cercului }
            Xstart, Ystart : integer ; { inceputul }
            Xend, Yend : integer; { sfirsitul }
        end;
VAR
    { Definire variabile }
    GraphGetMemPtr : pointer; { adresa memoriei pentru grafica }
    GraphFreeMemPtr : pointer; { adresa memoriei grafice libere }

```

3.3.9. Program demonstrativ — Exemple de utilizare

```

    {*****}
    { Exemple de utilizare a rutinelor grafice }
    { Biblioteca grafica Turbo Pascal versiunea 5.5 }
    {*****}
    { Autor: I. Nistor }

PROGRAM DEMONSTRATIE_CU_BIBLIOTECA_GRAFICA ;
uses graph,crt ;
var
    GraphDriver : integer ; {Driverul dispozitivului grafic }
    GraphMode : integer ; {Modul de lucru }
    MaxX,MaxY : integer ; {Rezolutia maxima a ecranului }
    CodEroare : integer ; {Codul de intoarcere al erorii de grafica }
    MaxCuloare : integer ; {Numarul maxim de culori disponibile }
    ch : char ;

```

Inițializare

```

{ EX01.Procedura de initializare }
PROCEDURE Initializare ;
begin GraphDriver :=Detect ; { Incarcare automata a
    { driver-ului pentru dispozitivul grafic disponibil}
    Initgraph(GraphDriver,GraphMode,'c/ ' ) ;
    CodEroare := GraphResult ;
    If CodEroare <> grOk then
        begin

```

```

Writeln('Eroare de grafica : ',
        GraphErrorMsg(CodEroare));
Halt(1) ;
end
else
begin
    MaxCuloare := GetMaxColor ;
    {Se obtine numarul maxim de culori disponibile }
    OutTextXY(230,160,'SINTETI IN MOD GRAFIC ') ;
    OutTextXY(130,190,'Apasati o tasta pentru a ');
    OutTextXY(130,210,' trece in mod alfanumeric');
    Repeat until keypressed ;
    {Pastreaza textul pe ecran pina se apasa }
    ch := ReadKey ; {pe o tasta oarecare }
    RestoreCrtMode ; {Se trece in mod alfanumeric }
    Writeln(' Sinteti in mod alfanumeric') ;
    Write(' Apasati o tasta pentru modul grafic');
    Repeat until keypressed ;
    ch := ReadKey ;
    SetGraphMode(GetGraphMode) ;
    { S-a revenit in mod grafic }
    OutTextXY(250,200,' DIN NOU ]N MOD GRAFIC ') ;
    Repeat until keypressed ;
    ch := ReadKey ;
end ;
ClearDevice ; {Sterge ecranul }
end ;

```

Funcție de conversie

```

{ EX02.Funcție de conversie }
FUNCTION Int2Str(L : LongInt) : string ;
{Realizeaza conversia intregului L intr-un
{ sir pentru a putea fi folosit de rutinele
{ OutText si OutTextXY }
var s : string ;
begin str(L,s) ;
    Int2Str :=s ;
end ;

```

Scrierea textelor

```

{ EX03.Desenare de texte - Fig.3-4 }
PROCEDURE Text ;
var font,marime : word ;
    w,h          : word ;
    ch           : char ;
    x,y          : integer ;
const marime_caractere : array[0..4] of word =(5,5,8,5,5) ;
begin
    OutTextXY(20,20,'Demonstratie cu SetTextJustify,

```

```

        SetTextStyle, SetUserCharSize');
{Fixare tipuri de caractere, marine, pozitie }
SetTextStyle(DefaultFont,VertDir,2);
SetTextJustify(CenterText,BottomText);      {Pozitionare text }
OutTextXY(20+TextWidth('M'),280,'VERTICAL');
        { Apelul functiei TextWidth }
SetTextStyle(DefaultFont,HorizDir,6) ;
SetTextJustify(LeftText,TopText) ;
OutTextXY(50+TextWidth('M'),80,'ORIZONTAL') ;
x := 320 ;
y := TextHeight('M') + 70;
For marine :=1 to 4 do
begin
    SetTextStyle(DefaultFont,HorizDir,marine) ;
    h := TextHeight('M') ; { Apel TextHeight }
    w := TextWidth('M') ;
    y := y +h ;
    OuttextXY(x,y,'Marime :' + Int2Str(marine));
                                {trasare numere intregi}
end ;
y := y + h div 2 ;
SetTextJustify(CenterText,TopText ) ;
SetUserCharSize(7,5,3,4) ; { Marimea caracterelor }
SetTextStyle(DefaultFont,HorizDir,UserCharSize) ;
OutTextYY(320,y+70,'Marime definita de utilizator');

```

Demonstratie cu SetTextJustify,SetTextStyle, SetUserCharSize

ORIZONTAL

Marime :1
Marime :2
Marime :3
Marime :4

Marime definita de utilizator

VERTICAL

Figura 3-4.

```

Repeat until keypressed ;
    ch := ReadKey ;
ClearDevice;

```

Tipurile de caractere

```
{EX04}
{Prezentarea formelor(fonturilor)de caractere - fig. 3-5}
SetTextJustify(LeftText,TopText);
SetTextStyle(DefaultFont,HorizDir,2);
OutTextXY(20,20,'TIPURI DE CARACTERE');
MoveTo(2,30); {Fixeaza punctul curent }
For font :=0 to 4 do
begin
  If font =3 then
  begin
    delay(6000);
    ClearDevice ;
  end;
  SetTextJustify(LeftText,TopText);
  MoveTo(2,GetY+40);
  { GetX,GetY obtin coordonatele punctului curent}
  If font =0 then
  begin
    SetTextStyle(font,Horizdir,1);
    ch:=#0 ;
    Repeat
    OutText(ch);
    If ( GetX + TextWidth('M')) > 720 then
    MoveTo(2,GetY + TextHeight('M')+3);
    ch :=succ(ch);
    until ch >=#255;
    end
  else
  begin
    SetTextStyle(font,HorizDir,marime_caractere[font]);
    ch :='!';
    Repeat
    Outtext(ch) ;
    If (GetX + TextWidth('M')) > 720 then
    MoveTo(2,GetY + TextHeight('M')+3) ;
    ch := Succ(ch) ;
    until (Ord(ch) =Ord('~') +1) ;
    end ;
  end ;
  Repeat until keypressed ;
  ch := readkey ;
  ClearDevice ;
end ;
```

Tipuri de linie predefinite

```
{ EX05      Fig 3-6 }
{ Prezentarea tipurilor de linie predefinite }
```

TIPURI DE CARACTERE

[illegible]

!"#\$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMN OPQRSTUVWXYZ
[\] ^ _ ` abcdefghijklmnopqrst
uvwxyz{|}~

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN  
OPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

!"#\$%&'()*+,-./0123456789:;<=
>?@ABCDEFGHIJKLMN O PQRSTU V
W X Y Z [\] ^ _ ' a b c d e f g h i j k l m n o p q r
s t u v w x y z { | } ~

!"#\$%&'()*+,-./0123456789:;<=>
?@ABCDEFGHIJKLMN O PQRSTU
VWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
{|}~

Figura 3-5.

```

{ Utilizatorul poate sa-si defineasca si alte tipuri de linie}
PROCEDURE TipLinie ;
var stil      : word ;
    pas       : word ;
    x,y       : integer ;
begin
    x := 85 ;
    y := 70 ;
    pas := 70 ;
    SetTextStyle(DefaultFont, HorizDir, 2) ;
    OutTextXY(170, 30, 'TIPURI DE LINIE PREDEFINITE') ;
    SetTextJustify(LeftText, TopText) ;
    OutTextXY(x, y, 'Latime normala ') ;
    SetTextJustify(CenterText, TopText) ;
    For stil := 0 to 3 do
    begin
        SetLineStyle(stil, 0, NormWidth) ;
        Line(x, y+50, x, y+180) ;
        SetTextStyle(DefaultFont, HorizDir, 1);
        OutTextXY(x, y+30, Int2Str(stil)) ;
        x := x + pas ;
    end ;
    SetTextJustify(LeftText, TopText);
    SetTextStyle(DefaultFont, HorizDir, 2);
    OutTextXY(x, y, 'Linii groase ') ;
    SetTextJustify(CenterText, TopText) ;
    For stil := 0 to 3 do
    begin
        SetLineStyle(stil, 0, ThickWidth) ;
        Line(x, y+50, x, y+180) ;
        SetTextStyle(DefaultFont, HorizDir, 1);
        OutTextXY(x, y+30, Int2Str(stil)) ;
        x := x + pas ;
    end ;
end ;

```

TIPURI DE LINIE PREDEFINITE

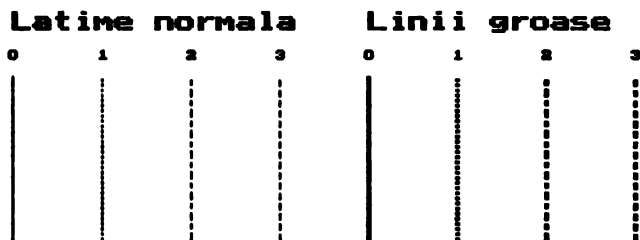


Figura 3-6.

```

Repeat until keypressed ;
ch := readkey ;
ClearDevice ;
end ;

```

Imagini deosebite

```

{ EX06 - Fig 3-7 }
{ Generare de imagini deosebite folosind PutPixel }
PROCEDURE Imagini_deosebite ;
var
  i,j,k      : integer ;
  s,x,y,xc,yc,r : real ;
begin
  xc := 11 ;
  yc := 11 ;
  s := 10 ;
  r := s/200 ;
  For i := 1 to 400 do
  begin
    x := xc + i * r ;
    For j := 1 to 200 Do
    begin
      y := yc + j * r ;
      k := trunc(x * y + x * y * y) ;
      {parte intreaga pentru x>0 }
      If k mod 2 = 0 then PUTPIXEL(I,J,1) ;
      {traseaza pixelul (i,j) in culoarea 1}
    end ;
  end ;
  Repeat until keypressed ;
  ch := readkey ;
  ClearDevice ;

end ;

```



Figura 3-7.

Diagramă circulară

```

{ EX07 }
{ Fig 3-8 } { Deseneaza o diagrama circulara }
PROCEDURE Diagrama_circulara ;
var xcentru   : integer ;
    ycentru   : integer ;
    raza      : word ;
    Xasp,Yasp : word ;
    x,y       : integer ;
PROCEDURE CoordonateText(unghi,raz : word ; var x,y : integer ) ;
{ se calculeaz{ coordonatele x,y ale capatului arcului }
var

```



```

radiani : real ;
begin
  radiani := unghi * pi / 180 ;
  {pi = 3.141592 }
  x := round(cos(radiani)*raza) ;
  y := round(sin(radiani)*raza) ;
end ;

```

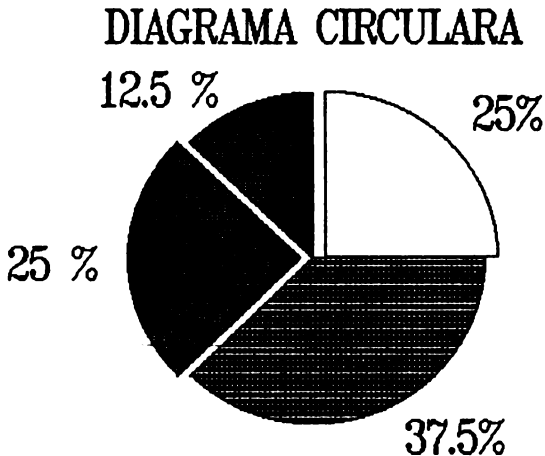


Figura 3-8.

```

begin
  GetAspectRatio(Xasp,Yasp); {calculeaza valorile Xasp si Yasp de
                              corectie a lui y pentru a nu deforma
                              cercurile si patratoarele }

  xcentru := 350 ;
  ycentru := 180 ; {Fixeaza centrul diagramei }
  raza := 166 ;
  while((Longint(raza)*Xasp) div Yasp) < round(200/3.6) do
    raza := raza +1 ;
  SetTextJustify(CenterText,TopText) ;
  SetTextStyle(TriplexFont,HorizDir,4) ;
  OutTextXY(320,20,'DIAGRAMA CIRCULARA') ;
  SetTextStyle(TriplexFont,HorizDir,Random(MaxCuloare)+1) ;
  {Fixare tip de hasurare si culoare }
  SetFillStyle(SolidFill,Random(MaxCuloare)+1);
  PieSlice(xcentru+10,ycentru-(10*Xasp) div Yasp,0,90,raza);
  {S-a trasat o parte din diagrama}
  CoordonateText(45,raza,x,y);
  SetTextJustify(LeftText,BottomText) ;
  OutTextXY(xcentru+10+X+TextWidth('H'),
            ycentru-(10+y)*Xasp div Yasp,'25%');
  SetFillStyle(HatchFill,Random(MaxCuloare)+1);
  PieSlice(xcentru,ycentru,225,360,raza);
  CoordonateText(293,raza,x,y);

```

```

SetTextJustify(LeftText,TopText);
OutTextXY(xcentru+x+TextWidth('h'),
          ycentru-(y*Xasp) div Yasp,'37.5%');
SetFillStyle(InterleaveFill,Random(MaxCuloare)+1);
PieSlice(xcentru-10,ycentru,135,225,raza);
CoordonateText(180,raza,x,y);
SetTextJustify(RightText,CenterText);
OutTextXY(xcentru-10+x-TextWidth('H'),
          ycentru-y*Xasp div Yasp,'25 %');
PieSlice(xcentru,ycentru,90,135,raza);
CoordonateText(112,raza,x,y);
SetTextJustify(RightText,BottomText);
OutTextXY(xcentru+x-TextWidth('H'),ycentru-y*Xasp div Yasp,'12.5 %');
Repeat until keypressed ;
  ch :=readkey ;
  ClearDevice;
end ;

```

Animație pe calculator

```

{ EX08 }
{Utilizarea rutinelor GetImage, PutImage, ImageSize,
  Rectangle, Ellipse, Circle }
PROCEDURE Animatie_pe_calculator ;
const raza = 20 ;
  xstart = 100 ;
  ystart = 50 ;
PROCEDURE Miscare(var x,y : integer ; Xpas,Ypas : integer ) ;
{Realizeaza miscarea obiectului }
var pas : integer ;
begin
  pas := Random(2*raza);{Calculeaza un numar aleator in (0,2*raza)}
  If Odd(pas) then
    {Functia Odd verifica daca pas este impar }
    pas := - pas;
  x := x +pas ;
  pas := Random(raza);
  If Odd(pas) then
    pas := - pas ;
  y := y + pas ;
  {Verifica daca obiectul iese in afara ecranului(vizorului)}
  If (x + xpas -1 > 600 ) then
    x := 600 - xpas + 1
  else
    If x < 25 then
      x := 25 ;
    If (y + ypas - 1 > getmaxy-25 ) then
      y := getmaxy-25 - ypas + 1
    else
      If y < 25 then
        y := 25 ;
  end ;
end ;

```

```

var timp_pauza : word ;
    sursa      : pointer ;
    x,y        : integer ;
    ulx,uly    : word ;
    lrx,lry    : word ;
    marime     : word ;
    i          : word ;

begin SetTextJustify(CenterText,TopText) ;
    SetTextStyle(TriplexFont,HorizDir,1) ;
    Rectangle(25,25,600,getmaxy-20);
    OutTextXY(320,1,'ANIMATIE PE CALCULATOR ' ) ;
    {Deseneaza obiectul care se misca }
    { Se deseneaza o farfurie zburatoare }
    Ellipse(xstart,ystart,0,360,raza,(raza div 3)+2);{ elipsa }
    Ellipse(xstart,ystart-4,190,357,raza,raza div 3) ;
    Line(xstart + 10,ystart - 6,xstart + 10,ystart - 12 ) ;
    Circle(xstart + 10,ystart -12,2) ; { Apel rutina CIRCLE }
    Line(xstart - 7,ystart - 6,xstart - 10,ystart - 12 ) ;
    Circle(xstart - 10,ystart -12,2);
    SetFillStyle(SolidFill,12);
    FloodFill(xstart + 1,ystart + 4,GetColor);
    { Citeste imaginea obiectului }
    ulx := xstart -(raza + 1) ;
    uly := ystart - 14 ;
    lrx := xstart + (raza + 1) ;
    lry := ystart + (raza div 3) + 3 ;
    marime := imageSize(ulx,uly,lrx,lry);
    { ImageSize determina numarul de pixeli cuprins intr-un dreptunghi }
    GetMem(sursa,marime); {Se alocă un spatiu de dimensiune "marime" }
                        { la adresa "sursa" }
    GetImage(ulx,uly,lrx,lry,sursa^); {Se incarca starea pixelilor din }
                        { dreptunghi la adresa "sursa" }
    PutImage(ulx,uly,sursa^,XORput);{Sterge imaginea obiectului }
    {Deseneaza citeva stele }
    For i:= 1 to 1000 do
        PutPixel(Random(580)+25,Random(getmaxy-50)+25,1) ;
        {Functia RANDOM(n) genereaza un numar aleator in intervalul (0,n) }
    x := 320 ;
    y := 99 ;
    { Pozitia initiala a farfuriei }
    timp_pauza := 70 ;
    { Urmeaza miscarea obiectului }
    Repeat
        PutImage(x,y,sursa^,XORput) ;{Deseneaza imaginea }
        Delay(timp_pauza ) ;
        PutImage(x,y,sursa^,XORput);{ Sterge imaginea }
        Miscare(x,y,lrx - ulx + 1,lry + uly + 1) ; { Deplaseaza imaginea }
    until keypressed ;
    ch := readkey ;
    ClearDevice ;
end ;

```

Diagramă dreptunghiulară

```
{ EX09 (Fig 3-9) -- Utilizarea rutinei BAR }
PROCEDURE Diagrama_dreptunghiulara ;
const nb = 5 ; { număr bare }
ib : array[1..nb] of byte =(1,3,5,2,4) ; { înălțimea barelor }
s : array[1..nb] of byte =(1,3,10,5,9) ; { stilul barelor }
var H      : word ;
Xpas,Ypas : real ;
i,j       : integer ;
c         : word ;
```

DIAGRAMA DREPTUNGIULARĂ



Figura 3-9.

```
begin
  H := 3 * TextHeight('M') ;
  SetTextJustify(CenterText,TopText) ;
  SetTextStyle(DefaultFont,HorizDir,2);
  OutTextXY(320,6,'DIAGRAMA DREPTUNGIULARA');
  SetTextStyle(DefaultFont,HorizDir,1);
  SetViewPort(50,30,590,470,ClipOn);
  Line(H,H+H,H,300-H); { AXA oy }
  Line(H,300-H,540-H,300-H);{ axa OX }
  Ypas := (300-3*H)/nb ;
  Xpas := (540-2*H)/nb ;
  J := 300-H ;
  SetTextJustify(CenterText,CenterText); { Marcheaza axa OY }
  For i := 0 to nb do
    begin
      Line(H div 2,j,H,j) ;
      OutTextXY(3,j,Int2Str(i)) ;
      j := round(j-Ypas) ;
    end ; { Marcheaza axa OX si deseneaza barele }
    j := H ;
    SetTextJustify(CenterText,TopText) ;
```

```

For i := 1 to nb+1 do
begin
SetColor(MaxCuloare);
Line(j,300-H,j,(300-35));
OutTextXY(j,300-12,Int2Str(i));
If i<> nb+1 then
begin
c:=Random(MaxCuloare)+1;
SetFillStyle(s[i],c);
SetColor(c);
Bar(j,round((300-H)-ib[i]*Ypas),round(j+Xpas),300-H-1);
Rectangle(j,round((300-H)-ib[i]*Ypas),round(j+Xpas),300-H-1);
end ;
j := round(j+Xpas) ;
end ;
Repeat until Keypressed ;
ch := readkey ;
ClearDevice ;
end ;

```

Haşurări

```

{ EX10 (Fig 3-10) -- Prezentarea tipurilor de hasura }
PROCEDURE Hasura ;
var
stil      : word ;
latime    : word ;
inaltime  : word ;
x,y       : word ;
I,J       : word ;
PROCEDURE Desen_bara(x,y : word) ;
begin
SetFillStyle(stil,MaxCuloare-1);
Bar(x,y,x+latime,y+inaltime);
Rectangle(x,y,x+latime,y+inaltime);
OutTextXY(x+(latime div 2),y+inaltime+8,Int2Str(stil));
stil := stil + 1;
end ;
begin
SetViewPort(0,0,639,479,ClipOn);
SetTextJustify(CenterText,CenterText);
SetTextStyle(DefaultFont,HorizDir,2);
OutTextXY(320,15,'TIPURI DE HASURA PREDEFINITA');
SetTextStyle(DefaultFont,HorizDir,1);
latime := 2*(641 div 13);
inaltime := 2*(330 div 10);
x := latime div 2;
y := inaltime div 2 ;
stil :=1 ;
For J := 1 to 3 do
begin
For I := 1 to 4 do

```

TIPURI DE HASURA PREDEFINITA

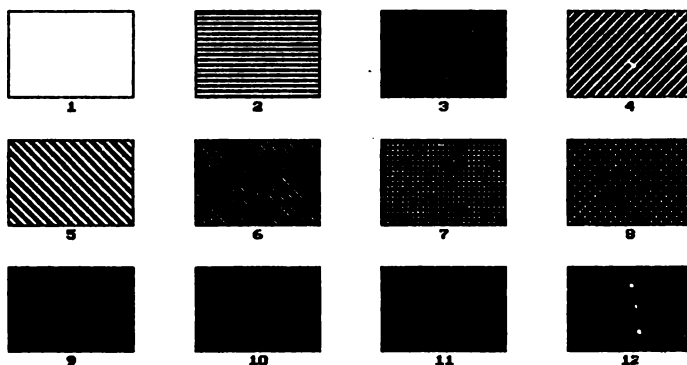


Figura 3-10.

```
begin
  Desen_bară(x,y);
  x := x + (latime div 2)*3;
end ;
x := latime div 2 ;
y := y + (inaltime div 2) * 3 ;
end ;
Repeat until keypressed ;
ch := readkey ;
ClearDevice ;
end ;
```

Paleta de culori

```
{EX11 -- Prezentarea paletii de culori disponibile }
PROCEDURE Demonstratie_culoare ;
var
  culoare   : word ;
  inaltime  : word ;
  latime    : word ;
  x,y       : word ;
  i,j       : word ;
PROCEDURE Desen_cutie(x,y : word ) ;
begin
  SetFillStyle(SolidFill,culoare);
  SetColor(culoare) ; {Fixeaza culoarea }
  Bar(x,y,x+latime,y+inaltime);
  Rectangle(x,y,x+latime,y+inaltime) ;
  culoare := GetColor ; { Obține culoarea }
  If culoare = 0 then
    begin
      SetColor(MaxCuloare);
```

```

    Rectangle(x,y,x+latime,y+inaltime) ;
  end ;
  OutTextXY(x+(latime div 2),y+inaltime+8,Int2Str(culoare));
  culoare := Succ(culoare) mod (MaxCuloare + 1) ;
end ;
begin
  SetTextJustify(CenterText,CenterText) ;
  SetTextStyle(DefaultFont,HorizDir,2);
  OutTextXY(320,15,'DEMONSTRATIE DE CULOARE');
  SetTextStyle(SmallFont,HorizDir,4);
  culoare := 1 ;
  latime := 2*(641 div 16) ;
  inaltime := 2*(330 div 10);
  x := latime div 2 ;
  y := inaltime div 2 ;
  For j := 1 to 3 do
    begin
      For i := 1 to 5 do
        begin
          Desen_cutie(x,y) ;
          x := x + (inaltime div 2 ) * 3 ;
        end ;
        x := latime div 2 ;
        y := y + (inaltime div 2 ) * 3 ;
      end ;
    Repeat until keypressed ;
    ch := readkey ;
    ClearDevice ;
  end ;
end ;

```

Programul principal

```

{*****}
{      PROGRAMUL PRINCIPAL      }
{*****}
{      Autor:  I. Nistor      }
begin
  Initializare ;
  Text ;
  TipLinie ;
  Imagini_deosebite ;
  Diagrama_circulara ;
  Animatie_pe_calculator ;
  Diagrama_dreptunghiulara ;
  Hasura ;
  Demonstratie_culoare ;
  CloseGraph ;
end.

```

3.3.10. Algoritmi și programe de grafică

ROTAȚIA UNUI TRIUNGHI ECHILATERAL

Fie (x_0, y_0) coordonatele originii sistemului cartezian de axe în raport cu originea spațiului de desen. Dacă vom considera un punct de coordonate (x_p, y_p) din spațiul de desen, atunci a determina coordonatele punctului obținut prin rotația spațiului imaginii în jurul punctului (x_0, y_0) și de unghi u , înseamnă a folosi formulele:

$$\begin{aligned}x_r &= x_0 + (x_p - x_0) * \cos(u) - (y_p - y_0) * \sin(u) \\y_r &= y_0 + (x_p - x_0) * \sin(u) + (y_p - y_0) * \cos(u)\end{aligned}$$

În cele ce urmează, fie (x_0, y_0) coordonatele punctului ce reprezintă centrul suprafeței de desen și r valoarea razei cercului în care vom înscrie un triunghi echilateral. Coordonatele vîrfulor triunghiului echilateral înscris în cercul considerat sînt următoarele:

$$\begin{aligned}x_1 &= x_0 + r, & y_1 &= y_0 \\x_2 &= x_0 - r / 2, & y_2 &= y_0 + r * \sqrt{3} / 2 \\x_3 &= x_2, & y_3 &= y_0 - r * \sqrt{3} / 2\end{aligned}$$

Pentru valori crescătoare ale unghiului de rotație u se va obține un șir de triunghiuri echilaterale cu proprietatea că fiecare triunghi are ca cerc înscris în el, cercul înscris în triunghiul inițial și că toate triunghiurile sînt înscrise în cercul circumscris în triunghiul inițial.

Programul care urmează implementează algoritmul prezentat mai sus pentru generarea șirului de triunghiuri echilaterale. Se vor citi de la monitor următoarele:

- r = raza cercului circumscris;
- p = pasul unghiului de rotație;
- n = numărul de rotații.

Programul ROTT

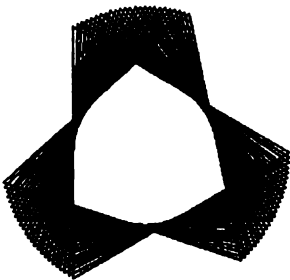
```
program ROTT; { Autori: Roxana Vlada, Marin Vlada }
{-----}
uses
  graph, crt ;
var
  graphdriver, graphmode      : integer ;
  r,p,x0,y0,x1,y1,x2,y2,x3,y3,f,fi : real   ;
  x1r,y1r,x2r,y2r,x3r,y3r      : real   ;
  n,i                          : integer  ;
  ch                           : char    ;
{-----}
begin { main }
  writeln(' ROTATIA unui triunghi echilateral ');
  write (' raza cercului circumscris r= ' ) ;
  read ( r ) ;
```



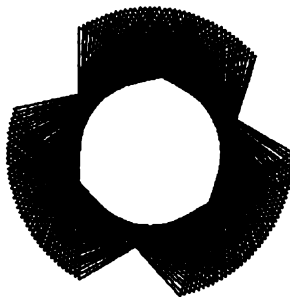
```

write ( ' pasul unghiului de rotatie p= ' ) ;
read ( p ) ;
write ( ' numarul de rotatii n = ' ) ;
read ( n ) ;
graphdriver := Detect ;
initgraph ( graphdriver, graphmode, ' ' ) ; { mod grafic }
setbkcolor ( 1 ) ; { culoare fond = albastru }
setcolor ( 4 ) ; { culoare pentru desen = rosu }
{ coordonatele cercului fata de originea (0,0) }
x0 := 300 ; y0 := 150 ;
f := pi / 180.0 ; { factor transformare in radiani }
{ coordonatele virfului triunghiului initial }
x1 := x0 + r ; y1 := y0 ;
x2 := x0 - r / 2.0 ; y2 := y0 + sqrt ( 3.0 ) * r / 2.0 ;
x3 := x2 ; y3 := y0 - sqrt ( 3.0 ) * r / 2.0 ;
{ generarea sirului de triunghiuri echilaterale }
for i := 0 to n - 1 do
begin
fi := f * p * i ;
{ coordonatele virfului triunghiului curent }
x1r := x0 + ( x1 - x0 ) * cos ( fi ) - ( y1 - y0 ) * sin ( fi ) ;
y1r := y0 + ( x1 - x0 ) * sin ( fi ) + ( y1 - y0 ) * cos ( fi ) ;
x2r := x0 + ( x2 - x0 ) * cos ( fi ) - ( y2 - y0 ) * sin ( fi ) ;
y2r := y0 + ( x2 - x0 ) * sin ( fi ) + ( y2 - y0 ) * cos ( fi ) ;
x3r := x0 + ( x3 - x0 ) * cos ( fi ) - ( y3 - y0 ) * sin ( fi ) ;
y3r := y0 + ( x3 - x0 ) * sin ( fi ) + ( y3 - y0 ) * cos ( fi ) ;
{ desenarea laturilor triunghiului curent }
moveto ( round(x1r), round(y1r) ) ; { pozitionare fara trasare }
lineto ( round(x2r), round(y2r) ) ; { trasare }
lineto ( round(x3r), round(y3r) ) ; { trasare }
lineto ( round(x1r), round(y1r) ) ; { trasare }
end ;
ch := readkey ; { citirea unui caracter oarecare }
closegraph ; { iesire mod grafic }
end.

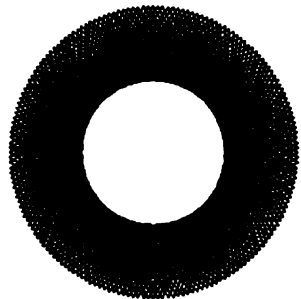
```



r=150, p=2, n=25



r=150, p=2, n=40



r=150, p=2, n=100

{=====}

Comentariu: Pentru valori diverse ale parametrilor p și n se obțin figuri care dau

impresia de suprafețe care se „frîng” (a se genera figuri pentru $n = 30, 50, 100, \dots$).

ROTAȚIA UNUI CERC

Fie (x_0, y_0) un punct avînd aceeași semnificație ca mai sus și r raza unui cerc dat de centru (x_0, y_0) . Vom genera un șir de cercuri de raza r și avînd centrele pe cercul de raza r și de centru (x_0, y_0) . Programul care implementează algoritmul de generare a cercurilor trebuie să aibă ca date de intrare următoarele elemente:

- (x_0, y_0) = coordonatele centrului;
- r = raza cercului inițial;
- p = pasul unghiului de rotație;
- n = numărul de rotații.

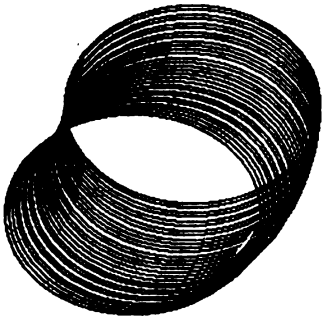
Programul ROTC

```
{=====}
program ROTC; { Autori: Roxana Vlada, Marin Vlada }
uses graph, crt ;
var
  graphdriver, graphmode : integer ;
  x0,y0,xr,yr,r,p,fi,f : real ;
  n,i : integer ;
  ch : char ;
{=====}
begin { main }
  writeln(' ROTAȚIA unui cerc ' ) ;
  write ( ' coordonatele centrului x0,y0 = ' ) ;
  read( x0,y0 ) ;
  write ( ' raza cercului initial r = ' ) ;
  read ( r ) ;
  write ( ' pasul unghiului de rotatie p = ' ) ;
  read ( p ) ;
  write ( ' numarul de rotatii n = ' ) ;
  read ( n ) ;
  graphdriver :=Detect ;
  initgraph ( graphdriver, graphmode, ' ' ) ; { mod grafic }
  setbkcolor ( 1 ) ; { fixare culoare fond - albastru }
  setcolor ( 4 ) ; { fixare culoare desen - rosu }
  f := pi / 180.0 ; { factor transformare in radiani }
  moveto ( round(x0), round(y0) ) ; { pozitionare fara trasare }
  { generarea sirului de cercuri }
  for i := 0 to n -1 do
    begin fi := f * p * i ;
      xr := x0 + r * cos ( fi ) ;
      yr := y0 + r * sin ( fi ) ;
      { desenarea cercului curent }
```

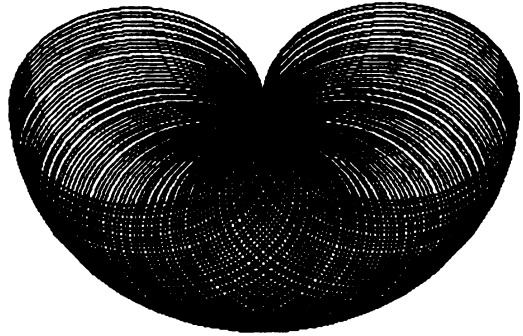
```

CIRCLE ( round ( xr ), round ( yr ), round ( r ) );
end;
ch := readkey ; { citirea unui caracter oarecare }
closegraph;      { iesire mod grafic }
end.

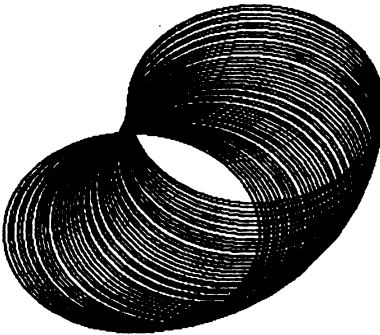
```



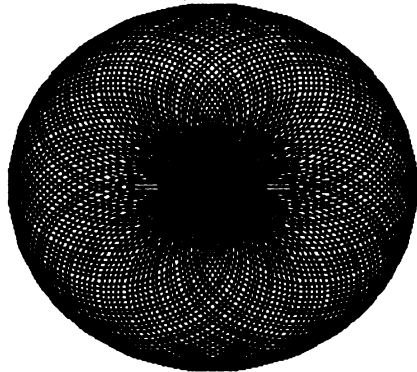
$r=100, p=2, n=30$



$r=100, p=2, n=90$



$r=100, p=2, n=45$



$r=80, p=3, n=200$

{-----}

Comentariu: Pentru diverse valori ale datelor de intrare se obțin figuri geometrice ce dau impresia de „corpuri rotunde” (a se genera desene pentru $n = 20, 50, 80, 100, \dots$).

EMBLEMA POLIGON

Fie $P = P_1 P_2 P_3 \dots P_n$ un poligon regulat cu n laturi, iar O centrul cercului circumscris acestui poligon. Pentru fiecare din cele n triunghiuri $O A_i A_{i+1}$, unde $i = \overline{1, n}$ (prin convenție $A_{n+1} = A_1$), se construiește un șir de triunghiuri conform următorului proces iterativ:

- fie ABC triunghiul inițial de pornire; se construiește triunghiul $A_1 B_1 C_1$ cu vîrfurile pe laturile triunghiului precedent, astfel ca

distanțele $d(A, A_1) = d(B, B_1) = d(C, C_1) = p$, unde p reprezintă lungimea unui segment de valoare suficient de mică, pe care o vom numi „pasul de construcție”;

- se continuă procesul de mai sus, obținându-se triunghiurile $A_i B_i C_i$, $i = 2, 3, \dots$ pînă cînd se va genera triunghiul $A_m B_m C_m$ cu proprietatea că $d(A_m, B_m) = l_{min}$, unde l_{min} este o valoare suficient de mică.

Pentru implementarea algoritmului schițat mai sus în scopul desenării celor n șiruri de triunghiuri, vom elabora procedura **LAT** care să calculeze coordonatele unui vîrf al triunghiului curent dintr-un șir dat, și procedura **FIG** care să construiască și să deseneze laturile triunghiurilor pentru cele n șiruri de triunghiuri.

Programul care desenează triunghiurile generate conform procesului de mai sus trebuie să citească de la monitor următoarele date:

- r = raza cercului circumscris poligonului;
- n = numărul de laturi ale poligonului;
- p = pasul de construcție a unui șir de triunghiuri;
- l_{min} = lungimea laturii triunghiului final.

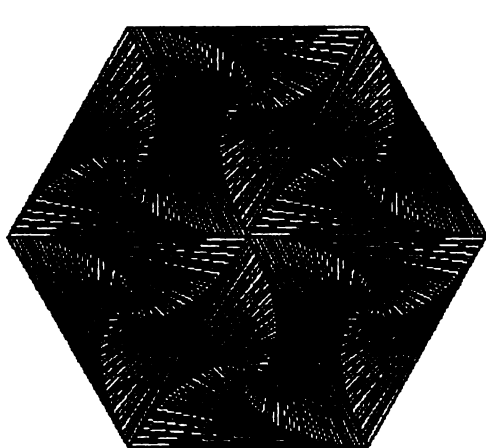
Programul POL

```
{-----}
program POL; { Autor: M. Vlada }
uses
  graph, crt ;
var
  graphdriver, graphmode      : integer ;
  ch                          : char   ;
  p, r, f, xa, ya, xb, yb, xc, yc, lmin : real   ;
  n, i                        : integer ;
{-----}
procedure LAT( p, xa, ya, xb, yb: real; var xl, yl: real);
{-----}
{ calculeaza coordonatele unui virf ptr. triunghiul curent }
var
  eps, fi, rap : real ;
begin
  eps := 1.0E -4 ;
  if ( abs(xa-xb) <= eps ) and ( yb < ya ) then
    fi:=3.0*pi/2.0;
  if ( abs(xa-xb) <= eps ) and ( yb > ya ) then
    fi:= pi / 2.0;
  if ( abs(xa-xb) > eps) and (yb >= ya ) then
    begin
      rap := ( yb-ya ) / (xb-xa) ;
      fi := arctan ( rap ) ;
```

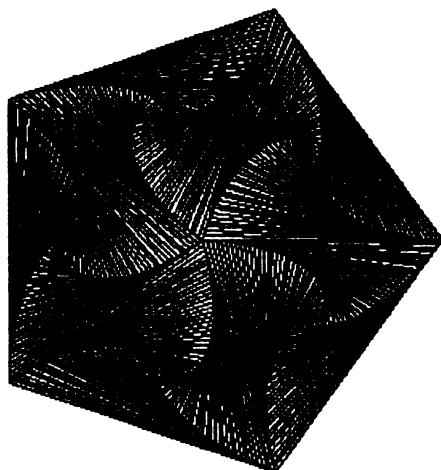
```

    if (abs(fi) <= eps) and (xb > xa) then
fi := 0.0;
    if fi < 0.0 then
fi := fi + pi ;
    end;
    if ( abs(xa-xb) > eps ) and ( yb < ya ) then
begin
    rap := ( yb-ya ) / (xb-xa) ;
    fi := arctan ( rap ) ;
    if fi > 0.0 then fi := fi + pi
        else fi := fi + 2*pi ;
    end;
    x1 := xa + p * cos ( fi ) ;
    y1 := ya + p * sin ( fi ) ;
end;
procedure FIG( p,xa,ya,xb,yb,xc,yc,lmin : real );
{ genereaza si traseaza un sir de triunghiuri }
var x1,y1,x2,y2,x3,y3,dist: real ;
begin
repeat
begin
    { coordonatele virfurilor triunghiului curent }
    LAT( p,xa,ya,xb,yb,x1,y1 ) ;
    moveto ( round ( x1 ), round ( y1 ) ) ; { fara trasare }
    LAT( p,xb,yb,xc,yc,x2,y2 ) ;
    lineto ( round ( x2 ), round ( y2 ) ) ; { trasare }
    LAT( p,xc,yc,xa,ya,x3,y3 ) ;
    lineto ( round ( x3 ), round ( y3 ) ) ; { trasare }
    lineto ( round ( x1 ), round ( y1 ) ) ; { trasare }
    { initializari ptr. reluarea procesului }
    xa := x1 ; ya := y1 ; xb := x2 ; yb := y2 ;
    xc := x3 ; yc := y3 ;
    dist := ( xa-xb ) * ( xa-xb ) + ( ya-yb ) * ( ya-yb ) ;
    dist := sqrt ( dist ) ;
end;
until abs( dist ) <= lmin ;
end;
begin { main }
    writeln( ' EMBLEMA poligon ( ARISTO ) ' );
    write ( ' raza cercului circumscris poligonului r = ' );
    read(r) ;
    write ( ' nr. de laturi ale poligonului n = ' ) ;
    read ( n ) ;
    write ( ' pasul de constructie p = ' ) ;
    read ( p ) ;
    write ( ' lungimea laturii triunghiului final lmin= ' ) ;
    read ( lmin ) ;
    f := pi / 180.0 ; { factor transformare in radiani }
    f := f * 360.0 / n ;
    graphdriver := Detect ;
    initgraph(graphdriver,graphmode,' ') ; { mod grafic }
    setviewport(320,170,500,200,false) ; { fixare origine }
    setbkcolor(1) ; { culoare fond = albastru }

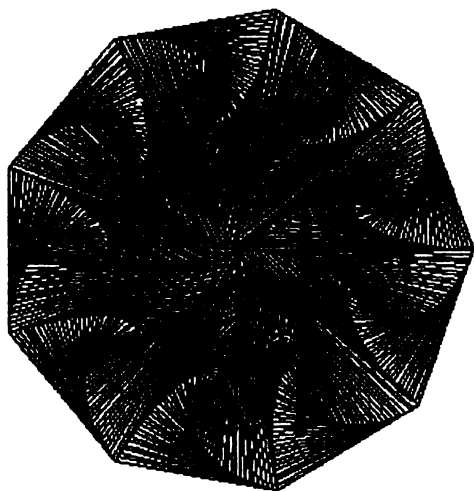
```



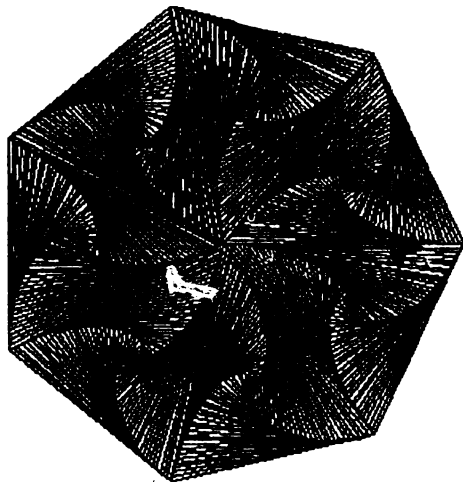
**$r=150, n=6,$
 $p=3, l=3$**



**$r=150, n=5,$
 $p=3, l=3$**



**$r=150, n=9,$
 $p=3, l=3$**



**$r=150, n=7,$
 $p=3, l=3$**

```

setcolor(4)                ;{ culoare desen = rosu      }
{ constructia celor n siruri de triunghiuri }
for i:= 1 to n do
begin
  { coordonatele triunghiului initial }
  xa := 0.0 ; ya := 0.0 ;
  xb := r * cos ( (i-1) * f ) ; yb := r * sin ( (i-1)*f ) ;
  xc := r * cos ( i * f )      ; yc := r * sin ( i * f ) .;
  moveto( round ( xa ), round ( ya ) ) ; { fara trasare }
  lineto( round ( xb ), round ( yb ) ) ; { trasare      }
  lineto( round ( xc ), round ( yc ) ) ;
  lineto( round ( xa ), round ( ya ) ) ;
  { generarea sirului de triunghiuri }
  FIG ( p,xa,ya,xb,yb,xc,yc,lmin) ;
end; ch := readkey; { citirea unui caracter oarecare }
closegraph ;      { iesire mod grafic }
end.
{-----}

```

Comentariu: *Figurile obținute prin procesul descris mai¹ sus dau impresia de suprafețe care se „înfășoară” (a se studia generări pentru $n=3..11$). Pentru $n=6$ figura generată este cunoscută sub numele de „emblemă ARISTO”.*

ÎNFĂȘURĂTOAREA POLIGON

Fie $P = P_1P_2P_3...P_n$ un poligon regulat cu n laturi și $Q = Q_1Q_2Q_3...Q_n$ poligonul regulat avînd vîrfurile pe laturile poligonului P , astfel că $d(P_1, Q_1) = d(P_2, Q_2) = d(P_3, Q_3) = ... = d(P_n, Q_n) = p$, unde p este valoarea unei lungimi suficient de mică, ce indică pasul de construcție. Dacă se continuă acest proces, se va obține un șir de poligoane regulate, fiecare poligon sprijinindu-se pe laturile poligonului precedent. Pentru implementarea algoritmului menționat vom elabora procedura **LAT** care calculează coordonatele vîrfurilor poligonului curent, și procedura **FIGP** care determină și desenează laturile șirului de poligoane. Programul trebuie să aibă la intrare următoarele elemente:

- r = raza cercului circumscris poligonului inițial;
- n = numărul de laturi ale unui poligon;
- p = pasul de construcție;
- $lmin$ = lungimea laturii poligonului final.

Programul POLR

```

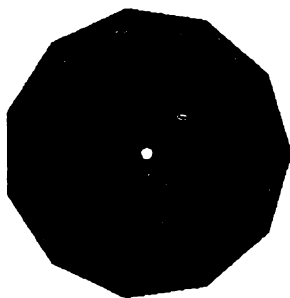
{-----}
program POLR; { Autori: F. Bristea, R. Niculescu }
uses graph, crt ;
var

```

```

graphdriver,graphmode : integer ;
ch                      : char   ;
p,r,f,fi,lmin          : real   ;
x, y, x1, y1           : array[1..100] of real ;
n,i                    : integer ;
=====
procedure LAT( p,xa,ya,xb,yb: real;
var x1,y1:real);
{ calculeaza coordonatele unui virf curent }
var eps, fi, rap : real ;
begin
  eps := 1.0E -4 ;
  if ( abs(xa-xb) <= eps ) and ( yb < ya ) then fi:=3.0*pi/2.0;
  if ( abs(xa-xb) <= eps ) and ( yb > ya ) then fi:= pi / 2.0;
  if ( abs(xa-xb) > eps) and (yb >= ya ) then
    begin
      rap := ( yb-ya) / (xb-xa) ;
      fi := arctan ( rap ) ;
      if (abs(fi) <= eps) and (xb > xa) then fi := 0.0;
      if fi < 0.0 then fi := fi + pi ;
    end;
  if ( abs(xa-xb) > eps ) and ( yb < ya ) then
    begin
      rap := ( yb-ya) / (xb-xa) ;
      fi := arctan ( rap ) ;
      if fi > 0.0 then fi := fi + pi
        else fi := fi + 2*pi ;
    end;
  x1 := xa + p * cos ( fi ) ;
  y1 := ya + p * sin ( fi ) ;
end;

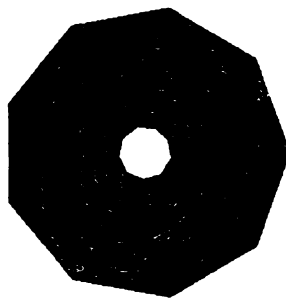
```



r=150, n=11,
p=4, l=4



r=150, n=11,
p=4, l=20



r=150, n=9,
p=4, l=20

```

procedure FIGP( p,lmin : real );
{-----}
var dist: real ;
begin
  repeat
    begin { determina si traseaza laturile poligonului curent }
      LAT( p,x[1],y[1],x[2],y[2],x1[1],y1[1] ) ;

```



```

moveto ( round ( x1[1] ), round ( y1[1] ) ) ; { fara trasare }
for i:= 2 to n do
  begin
    LAT( p,x[i],y[i],x[i+1],y[i+1],x1[i],y1[i] ) ;
    lineto( round( x1[i] ), round ( y1[i] ) ) ;
  end;
lineto( round ( x1[1] ), round ( y1[1] ) ) ;
x1[n+1] := x1[1] ; y1[n+1] := y1[1] ;
{ initializarea coordonatelor virfurilor poligonului }
{ pentru determinarea urmatorului poligon }
for i:= 1 to n + 1 do
  begin
    x[i] := x1[i] ; y[i] := y1[i] ;
  end;
dist := sqr(x[1]-x[2]) + sqr(y[1]-y[2]) ;
dist := sqrt ( dist ) ;
end;
until abs( dist ) <= lmin ;
end;
begin { main }
writeln( ' INFASURATOAREA poligon ( ARISTO ) ' );
write ( ' raza cercului circumscris poligonului r = ' ); read(r) ;
write ( ' nr. de laturi ale poligonului n = ' ) ; read ( n ) ;
write ( ' pasul de constructie p = ' ) ; read ( p ) ;
write ( ' lungimea laturii poligonului final lmin= ' ) ;
read ( lmin ) ;
f := pi / 180.0 ; { factor transformare in radiani }
f := f * 360.0 / n ;
graphdriver := Detect ;
initgraph(graphdriver,graphmode,' ' ) ; { mod grafic }
setviewport(320,170,500,200,false) ;{ fixare origine }
setbkcolor(1) ; { culoare fond = albastru }
setcolor(4) ; { culoare desen = rosu }
x[1] := r ; y[1] := 0.0 ; { initializare }
moveto ( round( r ), round ( 0.0 ) ) ; { pozitionare }
fi := 0.0 ; { initializare unghi }
{ trasarea poligonului regulat initial }
for i:= 1 to n do
  begin
    fi := fi + f ;
    x[i+1] := r * cos ( fi ) ; y[i+1] := r * sin ( fi ) ;
    lineto ( round ( x[i+1] ), round ( y[i+1] ) ) ; { trasare }
  end; { generarea si trasarea poligoanelor 2,3,... }
FIGP ( p,lmin) ;
ch := readkey; { citirea unui caracter oarecare }
closegraph ; { iesire mod grafic }
end.
{=====}

```

Comentariu: Acest program extinde procesul de construcție prezentat pentru un triunghi, la un poligon regulat oarecare. Pentru valori diverse ale parametrului n (de exemplu $n=3, 5, 7, 9, 11$) se obțin figuri deosebit de interesante.

CURBE GENERATE

În cele ce urmează vom considera o funcție periodică, de exemplu funcția $f: \mathbb{R} \rightarrow \mathbb{R}$, $f(t) = \sin(nt)$, unde n este un număr natural dat și vom scoate în evidență coordonatele polare (t, r) ale graficului dat de ecuația polară $r = f(t)$, adică $r = \sin(nt)$. Folosind reprezentarea grafică cu ajutorul calculatorului, se poate observa că graficul curbei de mai sus conține n petale, dacă n este impar și, $2n$ petale dacă n este par. Fie n și d numere naturale care să reprezinte valorile a două unghiuri exprimate în grade hexazecimale. Pentru $t = 0, d, 2d, \dots$, se consideră punctele date de coordonatele polare $(t, \sin(nt))$, care transformate în coordonate carteziene generează cuplul (x, y) , unde

$$\begin{aligned} x &= \sin(nt) \cdot \cos(t) \\ y &= \sin(nt) \cdot \sin(t). \end{aligned}$$

Prin urmare, se obțin șirurile de numere:

$$\begin{aligned} \{ t(m) \mid m \in \mathbb{N}, t(m) = m \cdot d \} \\ \{ x(m) \mid m \in \mathbb{N}, x(m) = \sin(n t(m)) \cdot \cos(t(m)) \} \\ \{ y(m) \mid m \in \mathbb{N}, y(m) = \sin(n t(m)) \cdot \sin(t(m)) \}. \end{aligned}$$

Considerând punctele P_m , $m=0,1,2,\dots$, având coordonatele $(x(m), y(m))$, puncte ce se află pe curba de ecuație polară $r = \sin(nt)$, dacă se unesc aceste puncte între ele, adică P_0 cu P_1 , P_1 cu P_2 și așa mai departe, pînă cînd se va uni ultimul punct generat (acest ultim punct va fi considerat relativ la grupul aditiv al claselor de resturi modulo 360), se va obține o construcție geometrică ce depinde de n, d și funcția periodică f considerată inițial.

TEOREMĂ: Fie G grupul aditiv al claselor de resturi modulo 360, atunci numărul de segmente care unesc punctele generate mai sus este egal cu ordinul elementului d în grupul G , adică este egal cu $360/k$, unde $k = \text{c.m.m.d.c.}(d, 360)$.

Figura generată astfel se obține utilizînd unghiurile corespunzătoare subgrupului $H = \langle d \rangle$ al grupului G (adică se consideră segmentele obținute prin unirea punctelor pentru $t = 0, 1, 2, \dots$). Subgrupul H generat de d are k clase de echivalență distincte și acestea sînt $H, H+1, H+2, \dots, H+k-1$.

În general, dacă $|H| < 360$, atunci se va obține un desen degenerat, și pentru a elimina această situație, se procedează la suprapunerea figurilor corespunzătoare claselor de echivalență $H+1, H+2, H+3, \dots$ pe desenul corespunzător clasei H . Pentru exemplificarea celor de mai sus și pentru înțelegerea modului de generare a figurilor, vom considera cazul particular $n=2$ și $d=90$. În acest caz, subgrupul H va conține 4 elemente și anume $0, d, 2d, 3d$ și astfel desenul corespunzător clasei H este format din 4 puncte

identice de coordonate $(0, 0)$, deci o figură degenerată. Pentru figura corespunzătoare clasei $H+1$, desenul generat este format din segmentele care se obțin prin unirea originii cu 4 puncte de pe curba de ecuație polară $r = \sin(2t)$ și procedeul se continuă și pentru clasele $H+2, H+3, H+4 \dots$. În acest caz, $k=4$ și pe același desen se suprapun desenele corespunzătoare claselor de echivalență $H, H+1, H+2, \dots, H+89$. Generarea desenului dă impresia că un pătrat își modifică dimensiunile astfel încât vîrfurile sale rămîn pe curba de ecuație polară $r = \sin(2t)$. Se poate observa că un pătrat, ce inițial este degenerat în origine, se mărește și în același timp se rotește puțin, după care începe să se micșoreze, continuînd pînă cînd se ajunge din nou în origine, deci s-a obținut din nou desenul corespunzător clasei H . În felul acesta vor fi generate 90 de pătrate.

Pentru $n=4$ și $d=90$ se va obține impresia de animație.

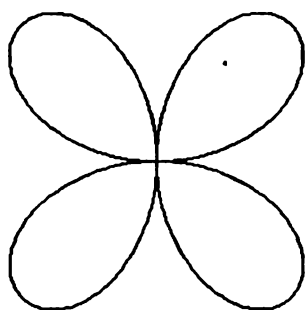
În algoritmul explicat mai sus s-a folosit grupul aditiv al claselor de resturi modulo 360, acest număr fiind măsura în grade a unui cerc complet. Algoritmul se poate aborda considerînd cazul mai general, și anume grupul aditiv al claselor de resturi modulo z , unde z este un număr natural între 1 și 360.

Programul care urmează implementează algoritmul în forma lui generală, și va avea la intrare următoarele:

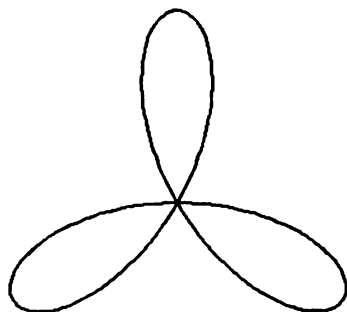
- z = unghiul pentru grupul claselor de resturi;
- n = numărul de petale;
- d = unghiul – perioadă.

Programul ROSE

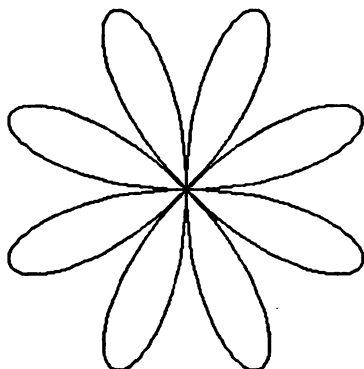
```
{=====}
program ROSE;    { Autori: M. Vlada, M. Garajeu }
uses
  graph,crt;
var
  graphdriver, graphmode      : integer;
  t, teta, fi, c              : integer;
  z, n, d                    : integer;
  x, a, Ax, Ay,s             : real   ;
  ch                          : char   ;
{=====}
begin
  writeln(' program < A rose is a rose... > ' );
  write  (' dati unghiul z=1,2,...,360 ? ' ) ; read( z ) ;
  write  (' dati nr. petale n z ? ' )       ; read ( n ) ;
  write  (' dati unghi-perioada d z ? ' )   ; read ( d ) ;
  t := 0; c := 0;
  graphdriver := Detect ;
  initgraph ( graphdriver, graphmode,' ' ) ; { init mod grafic }
  setviewport ( 300,150,500,200, false ) ; { fixare origine }
```



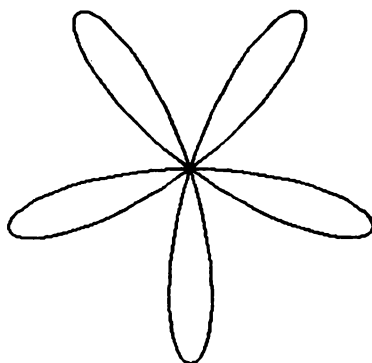
$n=2, d=1, z=360$



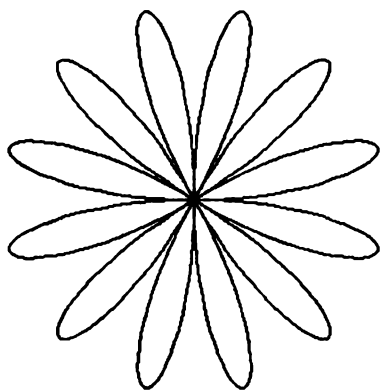
$n=3, d=1, z=360$



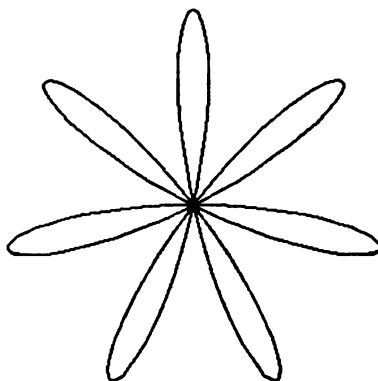
$n=4, d=1, z=360$



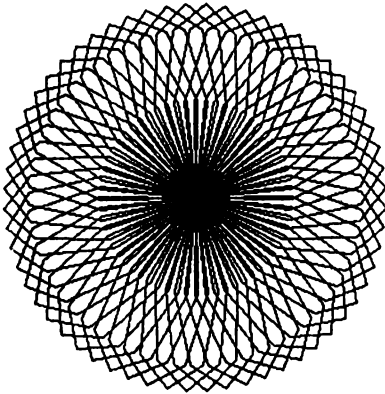
$n=5, d=1, z=360$



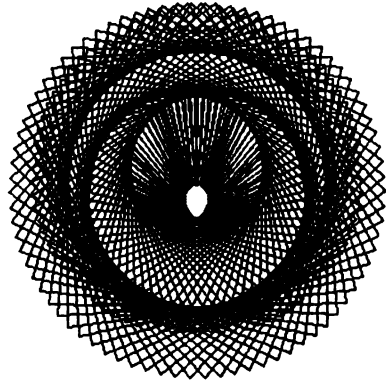
$n=6, d=1, z=360$



$n=7, d=1, z=360$

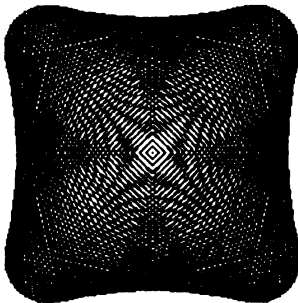


n=30, d=11, z=360

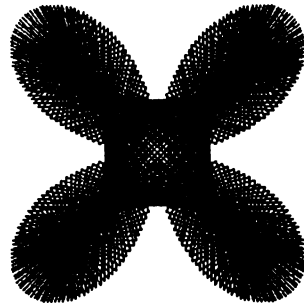


n=40, d=89, z=359

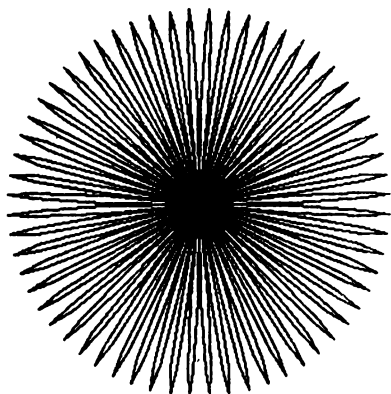
```
setcolor ( 4 ) ;      { fixare culoare desen = rosu }
setbkcolor ( 1 ) ; { fixare culoare fond = albastru }
repeat
begin
  teta := t ;
  a := t * 3.141592 / 180.0 ;    { transf. in coordonate polare }
  Ax := cos ( a ) * sin ( n * a ) * 150 ;
  Ay := sin ( a ) * sin ( n * a ) * 150 ;
  moveto ( round ( Ax ), round ( Ay ) ) ; { poz. fara trasare }
  repeat
  begin
    teta := teta + d ;
    teta := teta - round( teta / z ) * z ;
    fi := n * teta ;
    fi := fi - round( fi / z ) * z ;
    x := 2.0 * 3.141592 * fi / z ;
    s := 2.0 * 3.141592 * teta / z ;
```



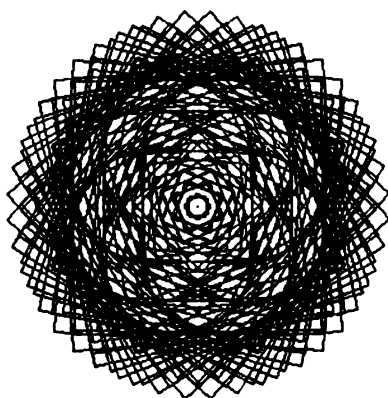
n=2, d=90, z=360



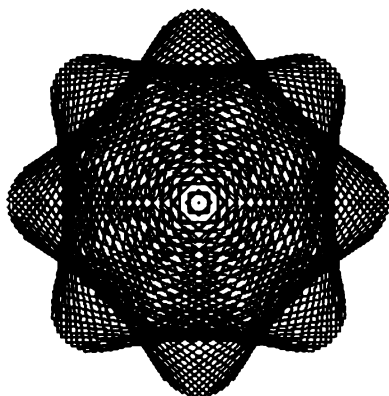
n=2, d=45, z=360



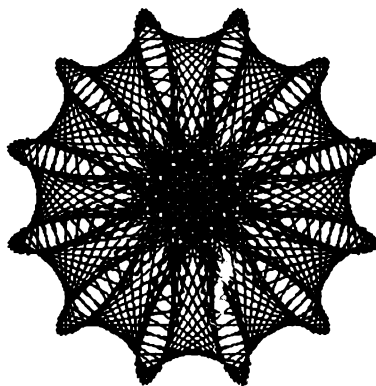
$n=30, d=1, z=360$



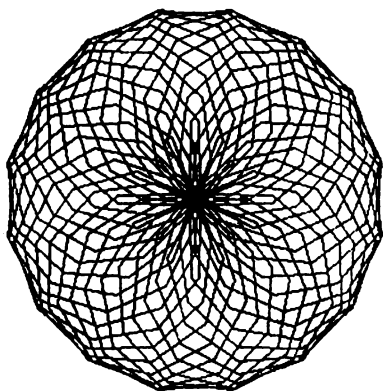
$n=68, d=90, z=360$



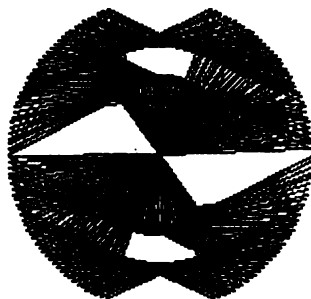
$n=88, d=90, z=360$



$n=6, d=140, z=360$



$n=8, d=160, z=360$



$n=900, d=90, z=360$

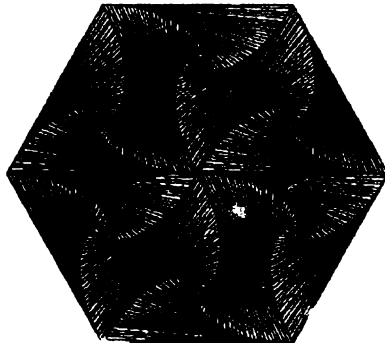
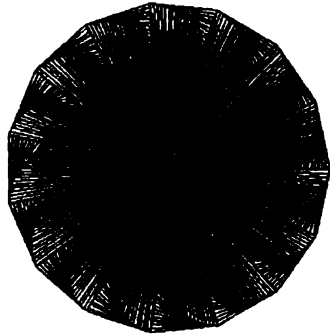
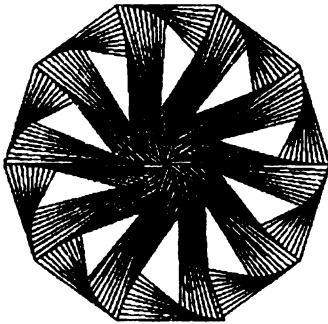
```

{ transformarea in coordonate carteziane }
Ax := sin ( x ) * cos ( s ) * 150 ;
Ay := sin ( x ) * sin ( s ) * 150 ;
{ trasare }
Lineto ( round ( Ax ), round ( Ay ) ) ;
c := c + 1 ;
end
until teta = t ;
t := t + 1 ;
end
until c >= z ;
repeat until keypressed ;
ch := readkey ; { inghetare desen pina la apasarea unei taste }
closegraph ; { iesire mod grafic }
end.
{-----}

```

Comentariu: Pentru diverse valori ale lui n și d se obțin figuri de o varietate și frumusețe deosebită. Pentru $(n, d) = (4, 120), (2, 90), (4, 90), (5, 97), (90, 90)$ sau $(6, 180)$, se obțin figuri complexe deoarece au loc procese de rotire a unor figuri cu vîrfuri pe anumite curbe, sau fenomene de micșorare sau mărire a diverselor figuri geometrice.

Notă: Cititorul poate face studiul de mai sus și pentru alte funcții periodice, de exemplu $f(t) = \sin(nt) + \cos(nt) - 1$, unde $n=1, 2, 3, \dots$.



3.4. Probleme propuse

Să se rezolve următoarele probleme folosind limbajul TURBO PASCAL:

- 1) Să se scrie un program care să rezolve ecuația:

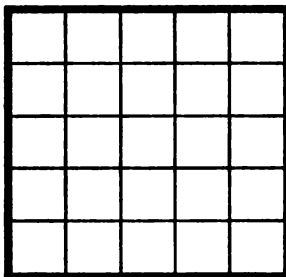
$$X_1 + X_2 + X_3 + \dots + X_n = m$$

unde $X_i \in \mathbb{N}$, $i=1, \dots, n$, $n < m$. Programul va prelua de la terminal valorile n și m . (Generalizarea unei probleme date la examenul de admitere în facultate, iulie 1989).

- 2) Se construiește o matrice 5×5 , conținând numerele de la 1 până la 25 astfel:

- poziția pe care este plasat 1 este citită de la intrare;
- dacă 1 ($0 < i < 25$) este plasat pe poziția de coordonate (x, y) , atunci numărul $i+1$ poate fi plasat numai pe una dintre următoarele poziții: $(x+3, y)$, $(x-3, y)$, $(x, y+3)$, $(x, y-3)$, $(x+2, y+2)$, $(x+2, y-2)$, $(x-2, y+2)$, $(x-2, y-2)$, fără a ieși în afara matricii.

(A) Date fiind la intrare coordonatele poziției în care apare 1, să se producă la ieșire o matrice construită conform regulilor de mai sus, cu respectarea încadrării în chenarul din figura următoare:



(B) Pentru fiecare alegere a poziției lui 1 în partea de sus dreaptă (inclusiv diagonală) a matricii, să se determine câte matrici de tipul considerat există.

(problemă dată la Olimpiada Internațională de Informatică a elevilor, Atena 1991)

- 3) Un s -termen este o secvență de 's', '(', ')', construită recursiv conform regulilor:

- s este un termen;
- dacă M și N sînt s -termeni, atunci și (MN) este un s -termen. Exemple: $((((ss)(ss))s)(ss))$ sau $((((ss(sss(ss$ (parantezele drepte putînd fi omise, neaducînd informații noi).

Lungimea unui s -termen este numărul de caractere 's' din el.

Se citește de la intrare numărul natural $n < 10$. Se cere să se scrie o procedură `gensterm` care să creeze pentru fiecare $k=1, \dots, n$ un fișier text care să conțină toți s -termenii de lungime k ; în fișier s -termenii sînt separați prin ';', iar fișierul se încheie cu '.

Se introduce asupra s -termenilor următoarea operație, numită reducere: orice s -subtermen avînd forma $((((sA)B)C))$ (unde A, B, C sînt s -subtermeni) poate fi transformat în $((AC)(BC))$, adică: $context1(((sA)B)C)context2 \rightarrow context1((AC)(BC))context2$.

Pentru simplificare vom spune în continuare „termen” în loc de „ s -termen”.

Pentru un termen dat există mai multe moduri în care poate fi ales un subtermen asupra căruia se poate efectua reducerea. Prin normalizarea unui termen se înțelege aplicarea succesivă a reducerii, atîta timp cît este posibil. Exemplu de șir de reduceri: $((((ss(sss(ss \rightarrow (((ss(((ss(ss \rightarrow ((s(ss(((ss(ss \rightarrow \dots \rightarrow ((s(ss((s(ss(s(ss$.

- Alegeți o structură de date adecvată pentru reprezentarea și reducerea termenilor. Scrieți două proceduri `readterm` și `printterm` care să treacă de la un termen la reprezentarea al căsă și invers. Demonstrați corectitudinea lor.
- Scrieți o procedură `reduce` care să efectueze o reducere a unui termen asupra

unui subtermen specificat și demonstrați corectitudinea ei.

- Scrieți o procedură **normalize** care, pentru un termen dat, efectuează succesiv reduceri pînă cînd nu mai este posibil sau pînă cînd numărul de reduceri depășește 30.
- Încorporați cele de mai sus într-un program care:
 - citește o valoare pentru n ,
 - utilizează termenii de lungime n , generați de **gensterm**,
 - transformă acești termeni în reprezentarea aleasă,
 - îi normalizează (dacă este posibil),
 - afișează rezultatul normalizării și numărul de reduceri efectuate pînă la normalizare sau nenormalizat dacă acest număr depășește 30,
 - afișează numărul de termeni de lungime n și numărul de termeni normalizați.

(problemă dată la Olimpiada Internațională de Informatică a elevilor, Atena 1991)

- 4) Să se scrie o rutină recursivă de căutare într-un arbore binar în care rădăcina este **ROOT**, iar fiecare nod este de forma (**legătură_dreapta**, **legătură_stînga**, **date**).
- 5) Să se obțină alte desene interesante schimbînd în procedura **Imagini_deosebite** (**EX06**) valoarea lui s sau funcția $f(xy) = xy + xy^2$. Indicații: $f(xy) = x^2 + y^2$, $f(xy) = xy + x$, etc.
- 6) Să se scrie un program care să rezolve ecuația lui Fermat în Z_m :

$$x^h + y^h = z^h, \quad h \in \mathbb{N}, h \geq 2.$$

Programul va prelua de la terminal valorile m și h . Să se reprezinte în dreptunghiul $[0, m] \times [0, m]$ punctele $(x, y) \in Z_m \times Z_m$ pentru care există $z \in Z_m$ astfel încît (x, y, z) să fie soluție a ecuației considerate.

- 7) Să se scrie un program de reprezentare grafică a unei funcții $f: [a, b] \rightarrow \mathbb{R}$, $a < b$. Funcția va fi dată printr-un subprogram de tip funcție, iar capetele intervalului a și b vor fi preluate de la terminal.
- 8) Se consideră o platformă dreptunghiulară avînd $m \times n$ pătrate. Pornind din pătratul de pe linia i și coloana j , un robot se poate deplasa în direcțiile **NORD**, **SUD**, **EST**, **VEST**. Mișcările robotului se execută la comenzi codificate prin caracterele **N**, **S**, **E**, **V**. La fiecare comandă robotul trece într-unul din pătratele vecine, iar dacă se încearcă părăsirea platformei, comanda respectivă se ignoră trecîndu-se la următoarea.

Dacă se citește de pe terminal valorile m, n, i, j și un șir de 60 de caractere-comandă, să se elaboreze un program pentru a reprezenta grafic platforma și drumul parcurs de robot în urma executării tuturor comenzilor, prezentîndu-se pozițiile prin care trece robotul cel puțin de două ori.

3.5. Anexe

3.5.1. Lista comenzilor MS-DOS V4.0

| Denumirea | Funcția |
|-----------|---|
| APPEND | Fixează o cale pentru fișierele de date |
| ASSIGN | Asignează o literă unui drive |
| ATTRIB | Setează ori fixează atributele fișierului |
| BACKUP | Salvează unul sau mai multe fișiere de pe un disc pe altul |
| BREAK | Setează controlul CTRL+C |
| CHCP | Afișează sau schimbă pagina cu codul curent pentru COMMAND.COM |
| CHDIR(CD) | Schimbă directorul sau listează directorul de lucru |
| CHKDSK | Scanează discul din drive-ul specificat și verifică dacă există erori |
| CLS | Șterge ecranul |
| COMMAND | Startează procesorul de comenzi |
| COMP | Compară conținutul a două seturi de fișiere |
| COPY | Copiază fișierele specificate |
| CTTY | Permite schimbarea unității de la care se dau comenzile |
| DATE | Afișează sau setează data |
| DEL | Șterge fișierele specificate |
| DIR | Listează fișierele din director |
| DISKCOMP | Compară conținutul a două discuri |
| DISKCOPY | Copiază conținutul unui disc pe alt disc |
| EXE2BIN | Realizează conversia fișierelor executabile (.exe) în format binar |
| EXIT | Ieșirea din procesorul de comenzi și întoarcerea la nivelul anterior (dacă există) |
| FASTOPEN | Crește performanțele sistemului în închiderea și deschiderea fișierelor (indicat în aplicațiile cu multe fișiere) |

| Denumirea | Funcția |
|------------|---|
| FC | Compară fișiere și afișează diferențele între ele |
| FDISK | Configurează harddiscul pentru MS-DOS |
| FIND | Caută un șir într-un text |
| FORMAT | Formatează un disc |
| GRAPHTABL | Încarcă un tabel cu caractere grafice |
| GRAPHICS | Pregătește MS-DOS-ul pentru tipărirea grafică |
| JOIN | Atașează un drive la o cale |
| KEYB | Încarcă un program de la tastatură |
| LABEL | Etichetează discurile |
| MEM | Afișează informații despre memorie |
| MKDIR(MD) | Creează un director |
| MODE | Setează modulele de operare pentru unități |
| MORE | Afișează ieșirile pe ecran la un moment dat |
| NLSFUNC | Încarcă informații specifice unei țări |
| PATH | Setează o cale de căutare |
| PRINT | Tipărește fișiere |
| PROMPT | Schimbă prompter-ul |
| RECOVER | Reface un disc (fișier) defect |
| REN | Redefinește un fișier |
| REPLACE | Înlocuiește un fișier cu un alt fișier |
| RESTORE | Restaurează fișierele salvate |
| RMDIR (RD) | Șterge un director |
| SELECT | Instalează MS-DOS-ul pe un nou disc cu informațiile specifice țării și tastaturii |
| SET | Permite înlocuirea unui șir cu un alt șir |
| SHARE | Instalează partajarea fișierelor și blocarea |
| SORT | Sortează date ascendent sau descendent |

| Denumirea | Funcția |
|-----------|---|
| SUBST | Permite asocierea unei litere pentru o cale. Se crează un drive virtual |
| SYS | Transferă fișierele sistem MS-DOS dintr-un drive în altul |
| TIME | Furnizează sau setează timpul |
| TREE | Afișează directoarele și numele fișierelor |
| TYPE | Afișează conținutul unui fișier |
| VER | Afișează numărul versiunii sistemului MS-DOS |
| VERIFY | Verifică scrierile în disc |
| VOL | Afișează etichetele volumelor |
| XCOPY | Copiază fișiere și subdirectoare |

3.5.2. Cuvinte rezervate în TURBO PASCAL versiunea 5.5

| | | |
|--------------------|-----------------------|----------------|
| absolute | goto | repeat |
| and | if | set |
| array | implementation | shl |
| begin | in | shr |
| case | inline | string |
| const | interface | then |
| constructor | interrupt | to |
| destructor | label | type |
| div | mod | unit |
| do | nil | until |
| downto | not | uses |
| else | object | var |
| end | of | virtual |
| external | or | while |
| file | packed | with |
| for | procedure | xor |
| forward | program | |
| function | record | |

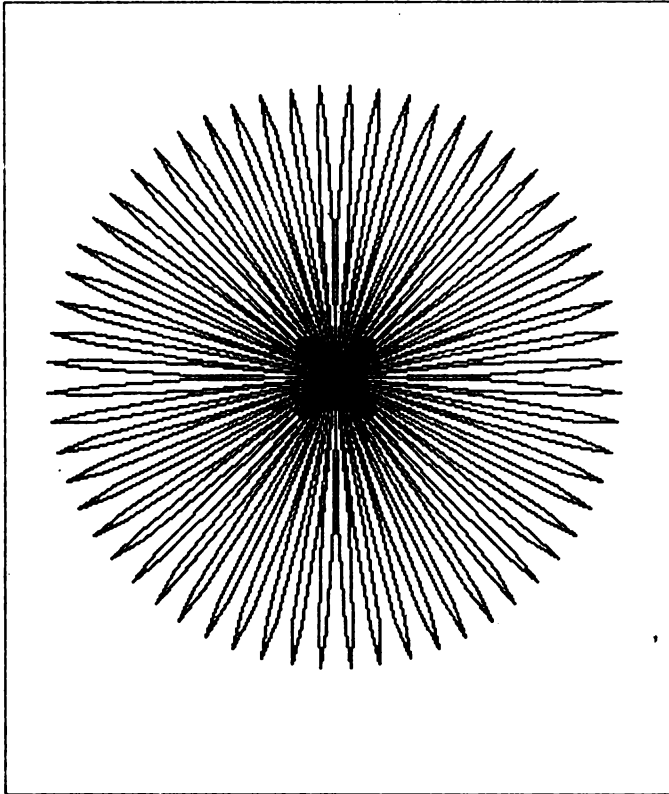
3.5.3. Proceduri și funcții în biblioteca grafică GRAPH.TPU

| | |
|-------------------|-------------------|
| Arc | FillPoly |
| Bar | FloodFill |
| Bar3D | GetArcCoords |
| Circle | GetAspectRatio |
| ClearDevice | GetBkColor |
| ClearViewport | GetColor |
| CloseGraph | GetDefaultPalette |
| DetectGraph | GetDriverName |
| Drawpoly | GetFillPattern |
| Ellipse | GetFillSettings |
| FillEllipse | GetGraphMode |
| GetImage | GraphDefaults |
| GetLineSettings | GraphErrorMsg |
| GetMaxMode | GraphResult |
| GetMaxColor | ImageSize |
| GetMaxX | InitGraph |
| GetMaxY | InstallUserDriver |
| GetModeName | InstallUserFont |
| GetModeRange | Line |
| GetPalette | LineRel |
| GetPaletteSize | LineTo |
| GetPixel | MoveRel |
| GetTextSettings | MoveTo |
| GetViewSettings | OutText |
| GetX | OutTextXY |
| GetY | PieSlice |
| PutImage | PutPixel |
| Rectangle | SetGraphMode |
| RegisterBGIdriver | SetLineStyle |
| RegisterBGIfont | SetPalette |
| RestoreCrtMode | SetRGBPalette |
| Sector | SetTextJustify |
| SetActivePage | SetTextStyle |
| SetAllPalette | SetUserCharSize |
| SetAspectRatio | SetViewport |
| SetBkColor | SetVisualPage |
| SetColor | SetWriteMode |
| SetFillPattern | TextHeight |
| SetFillStyle | TextWidth |

3.6. Bibliografie

- [1] * * * *MS-DOS Operating System, User's Guide*, Microsoft Corp.
- [2] * * * *MS-DOS Operating System, User's Reference*, Microsoft Corp.
- [3] * * * *TURBO PASCAL Owner's Handbook*, Borland International.
- [4] Uwe Beck
Computer – Graphik, Bilder und Programme zu Fraktalen, Chaos und Selbstähnlichkeit
Birkhäuser Verlag, 1988.
- [5] R. Halpen
Microcomputer Graphics Using PASCAL, Harper Row, Publishers, New York, 1985.
- [6] M. Vlada, A. Posea
Grafică automată în limbajul FORTRAN 77 și aplicații, Tipografia Universității din București, 1990.

INTERPOLAREA CURBELOR PLANE



| | |
|---|------------|
| 4. INTERPOLAREA CURBELOR PLANE | 173 |
| 4.1. Interpolarea cu funcții spline cubice | 176 |
| 4.2. Interpretarea fizică a interpolării cu funcții spline | 180 |
| 4.3. Considerații de implementare | 181 |
| 4.4. Determinarea analitică a parametrilor curbei | 183 |
| 4.5. Calculul numeric al parametrilor curbei | 186 |
| 4.6. Interpolare cu un număr redus de puncte | 190 |
| 4.7. Trasarea curbelor plane definite parametric | 196 |
| 4.8. Algoritm de trasare eficientă | 201 |
| 4.9. Bibliografie | 209 |

In literatura de specialitate din domeniul graficii pe calculator se întâlnesc din abundență lucrări și articole pe tema interpolării și aproximării curbelor plane, adică a construirii complete a unei curbe continue pornind de la o mulțime finită de puncte cunoscute inițial. Aceste lucrări conțin de obicei o prezentare riguroasă a suportului matematic al problemei interpolării și furnizează indicații generale de implementare, ajungând câteodată la prezentarea unor algoritmi de principiu [1]. Acești algoritmi se dovedesc în general puțin folositori sau ineficienți la o abordare concretă a problemei interpolării, în timp ce algoritmi detaliați și eficienți de implementare a interpolării sînt greu de găsit în literatură. De exemplu, problema generării unei linii drepte care unește două puncte date pe un display grafic cu rastru dreptunghiular, este rezolvată concret și eficient prin binecunoscutul algoritm de generare al lui J. Bresenham; ceva asemănător pentru rezolvarea problemei interpolării nu se întâlnește în literatură — e drept însă, această problemă este mult mai complexă.

În cele ce urmează vom prezenta problemele practice pe care le considerăm cele mai importante, întâlnite la o încercare de implementare concretă a interpolării și trasării curbelor plane. Vom acorda atenție specială chestiunilor legate de calculele aritmetice efective necesare pentru implementarea algoritmilor — evitarea depășirilor aritmetice, limitarea erorilor de rotunjire cumulative pentru valorile calculate iterativ, etc. De asemenea, vom urmări optimizarea algoritmilor prezentați în privința eficienței (vitezei) de execuție; în acest scop vom căuta să micșorăm numărul operațiilor în virgulă mobilă, în favoarea celor cu numere întregi (virgulă fixă) și vom urmări reducerea numărului de înmulțiri și împărțiri. Aceste aspecte sînt deosebit de importante pentru o implementare concretă a interpolării pe un dispozitiv periferic de afișare grafică (display cu rastru rectangular, plotter) bazat pe microprocesor: microprocesoarele uzuale nu conțin aritmetică în virgulă mobilă, iar instrucțiunile de înmulțire și împărțire în virgulă fixă sînt implementate în general numai pe procesoare de 16 biți și sînt mult mai puțin eficiente decît operațiile de adunare și scădere.

Secțiunea 4.1 conține o prezentare teoretică a problemei interpolării cu funcții spline cubice pe porțiuni, iar în secțiunea 4.2 se demonstrează

o proprietate extremală a acestor funcții; concluzia este că interpolarea cu funcții spline cubice dă rezultatele cele mai naturale și că acest tip de interpolare este cel mai indicat pentru modelarea fenomenelor reale.

Pentru implementarea practică a unui mecanism de interpolare trebuie rezolvate două probleme majore și principial distincte:

- calcularea parametrilor de interpolare, care definesc expresia analitică a curbei care trebuie trasată;
- trasarea efectivă a curbei interpolatoare.

În secțiunea 4.4 este prezentată o metodă teoretică de calcul al parametrilor de interpolare, deducându-se expresii analitice pentru acești parametri în funcție de coordonatele punctelor fixate inițial; în secțiunea 4.5 este prezentată o procedură de calcul numeric aproximativ al acestora. Se dovedește că evaluarea parametrilor de interpolare nu ridică probleme de calcul deosebite, obținându-se rezultate suficient de precise chiar în cazul utilizării exclusive a aritmeticii în virgulă fixă.

Secțiunea 4.6 descrie un caz special de interpolare, de mare interes practic pentru aplicațiile interactive, când trasarea curbei trebuie începută înainte de definirea tuturor punctelor fixe.

În secțiunea 4.7 este discutată în general problema trasării curbelor plane definite parametric, pe un dispozitiv de afișare grafică cu rezoluție finită. Este prezentată o procedură de trasare a curbelor plane, bazată pe metoda înjumătățirii intervalului de variație a parametrului independent din expresia analitică a curbei.

În cazul interpolării cu funcții spline cubice, curbele care trebuie trasate sînt reprezentate parametric prin polinoame de gradul 3 (sau mai mic). În acest caz particular, algoritmul de trasare prin metoda înjumătățirii poate fi optimizat, astfel încît să nu utilizeze înmulțiri și împărțiri, ci numai operații simple de adunare, scădere și deplasare binară. Acest rezultat, obținut în secțiunea 4.8, este foarte important din punct de vedere al implementării; aceasta se datorează faptului că eficiența procedurii de trasare este determinantă pentru performanțele globale ale programului de interpolare cu vizualizare.

4.1. Interpolarea cu funcții spline cubice

Problema teoretică clasică a interpolării funcțiilor după date exacte, definite pe o mulțime discretă de puncte, este rezolvată cu ajutorul polinoamelor interpolative ale lui Lagrange. Această metodă asociază unui set de n puncte din plan, de abscise diferite, o curbă polinomială de gradul $n-1$ care trece prin toate cele n puncte. Metoda are avantajul continuității tuturor derivatelor curbei interpolatoare. Totuși, curba interpolatoare Lagrange nu este naturală în sensul că oscilează prea mult

între punctele fixe; acest aspect este prohibitiv pentru dispozitivele grafice care utilizează operații de interpolare pentru modelarea unor fenomene naturale.

În ultimii ani s-au impus pe plan teoretic și mai ales practic metode noi și eficiente de interpolare, care au primit denumirea de interpolări cu *funcții spline*. O funcție spline este de obicei polinomială pe porțiuni, adică o funcție pentru care există o diviziune a intervalului I de interpolare în subintervale, astfel încât în interiorul fiecărui subinterval al diviziunii funcția reprezintă un polinom de un anumit grad m . În plus, funcția este de regulă continuă pe I împreună cu derivatele sale până la ordinul $m-1$ inclusiv, și are derivata de ordinul m cu pătrat integrabil.

Cele mai utilizate funcții spline s-au dovedit a fi polinoamele de gradul trei. În acest caz, curba interpolatoare este continuă împreună cu primele două derivate, ceea ce este suficient pentru modelarea fenomenelor naturale. În plus, după cum se va dovedi în secțiunea 4.2, curba interpolatoare cu funcții spline cubice are o proprietate extremală naturală care minimizează oscilația și asigură pentru curbă aspectul cel mai estetic, natural cu putință. În continuare, vom prezenta pe scurt problema teoretică a interpolării cubice pe porțiuni pentru funcții de o variabilă.

Fie o rețea de puncte pe dreapta reală $t_0 < t_1 < t_2 < \dots < t_n$ în ale cărei noduri sînt date valorile $\{f_i\}$, $i = \overline{0, n}$ ale funcției $f: [t_0, t_n] \rightarrow \mathbb{R}$. Se pune problema găsirii unei funcții $g: [t_0, t_n] \rightarrow \mathbb{R}$, numită *funcție interpolatoare*, care să îndeplinească următoarele condiții:

- $g(t)$ este de clasă $C_{(t_0, t_n)}^{(2)}$, adică este continuă pe (t_0, t_n) împreună cu derivatele sale de ordinele întâi și doi;
- pe fiecare interval $[t_i, t_{i+1}]$ funcția $g(t)$ este un polinom de gradul 3;
- în nodurile rețelei $\{t_i\}$, $i = \overline{0, n}$ sînt îndeplinite egalitățile:

$$g(t_i) = f_i, i = \overline{0, n}; \quad (1.1)$$

- $g''(t)$ satisface condițiile la limită:

$$g''(t_0) = g''(t_n) = 0. \quad (1.2)$$

Vom arăta că în condițiile de mai sus problema găsirii funcției interpolatoare cubice pe porțiuni $g(t)$ are soluție unică.

Deoarece derivata a doua a funcției $g(t)$ este liniară pe fiecare interval $[t_i, t_{i+1}]$, $i = \overline{0, n-1}$ al rețelei, putem scrie pentru $t_i \leq t \leq t_{i+1}$:

$$g''(t) = 6a_i \frac{t_{i+1}-t}{d_i} + 6b_i \frac{t-t_i}{d_i}, \quad (1.3)$$

unde am notat $d_i = t_{i+1} - t_i$, $6a_i = g''(t_i + 0)$, $6b_i = g''(t_{i+1} - 0)$, pentru $i = \overline{0, n-1}$. Din condiția de continuitate a derivatei a doua în punctele

$t_i, i = \overline{1, n-1}$, obținem imediat:

$$b_i = a_{i+1}, i = \overline{0, n-2}, \quad (1.4)$$

egalități pe care le completăm definind $a_n = b_{n-1} = 0$, conform (1.2).

Integrând de două ori ambii membri ai egalității (1.3), obținem pentru $t_i \leq t \leq t_{i+1}$:

$$g(t) = a_i \frac{(t_{i+1}-t)^3}{d_i} + a_{i+1} \frac{(t-t_i)^3}{d_i} + A_i \frac{t_{i+1}-t}{d_i} + B_i \frac{t-t_i}{d_i} \quad (1.5)$$

unde A_i și B_i sînt constante de integrare care se calculează din condițiile de interpolare (1.1) $g(t_i) = f_i$, $g(t_{i+1}) = f_{i+1}$. Introducînd $t = t_i$ și $t = t_{i+1}$ în (1.5), obținem succesiv:

$$a_i d_i^2 + A_i = f_i, \quad a_{i+1} d_i^2 + B_i = f_{i+1}$$

$$g(t) = a_i \frac{(t_{i+1}-t)^3}{d_i} + a_{i+1} \frac{(t-t_i)^3}{d_i} + (f_i - a_i d_i^2) \frac{t_{i+1}-t}{d_i} + (f_{i+1} - a_{i+1} d_i^2) \frac{t-t_i}{d_i},$$

$$t_i \leq t \leq t_{i+1}, i = \overline{0, n-1} \quad (1.6)$$

$$g'(t) = -3a_i \frac{(t_{i+1}-t)^2}{d_i} + 3a_{i+1} \frac{(t-t_i)^2}{d_i} + \frac{f_{i+1}-f_i}{d_i} - (a_{i+1}-a_i) d_i,$$

$$t_i \leq t \leq t_{i+1}. \quad (1.7)$$

În (1.7) aplicăm condiția de continuitate a derivatei în t_i egalînd derivatele laterale în punctele $t_i, i = \overline{1, n-1}$:

$$g'(t_i + 0) = -3a_i d_i + \frac{f_{i+1}-f_i}{d_i} - (a_{i+1}-a_i) d_i =$$

$$= -2a_i d_i - a_{i+1} d_i + \frac{f_{i+1}-f_i}{d_i}, i = \overline{0, n-1}$$

$$g'(t_{i+1} - 0) = 3a_{i+1} d_i + \frac{f_{i+1}-f_i}{d_i} - (a_{i+1}-a_i) d_i =$$

$$= 2a_{i+1} d_i + a_i d_i + \frac{f_{i+1}-f_i}{d_i}, i = \overline{0, n-1}$$

$$g'(t_i - 0) = 2a_i d_{i-1} + a_{i-1} d_{i-1} + \frac{f_i-f_{i-1}}{d_{i-1}}, i = \overline{1, n}$$

$$d_{i-1} a_{i-1} + 2(d_{i-1} + d_i) a_i + d_i a_{i+1} = \frac{f_{i+1}-f_i}{d_i} - \frac{f_i-f_{i-1}}{d_{i-1}}, i = \overline{1, n-1}. \quad (1.8)$$

Completînd ecuațiile (1.8) cu egalitățile $a_0 = a_n = 0$ rezultate din condițiile la limită (1.2), obținem un sistem liniar de $n-1$ ecuații cu $n-1$

necunoscute a_1, a_2, \dots, a_{n-1} . Matricea sistemului este:

$$M = \begin{pmatrix} 2(d_0+d_1) & d_1 & 0 & 0 & \dots & 0 & 0 \\ d_1 & 2(d_1+d_2) & d_2 & 0 & \dots & 0 & 0 \\ 0 & d_2 & 2(d_2+d_3) & d_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & d_{n-2} & 2(d_{n-2}+d_{n-1}) \end{pmatrix}$$

Matricea M este tridiagonală simetrică, cu dominantă diagonală strictă. Se arată ușor că această matrice este pozitiv definită și prin urmare nesingulară. Într-adevăr, să evaluăm forma pătratică $\sum_{i,j} m_{ij} x_i x_j$, unde

$M = (m_{ij})$ iar numerele x_i nu sînt toate nule. Obținem:

$$\begin{aligned} \sum_{i,j=1}^{n-1} m_{ij} x_i x_j &= \sum_{i=1}^{n-1} 2(d_{i-1}+d_i) x_i^2 + \sum_{i=1}^{n-2} d_i x_{i+1} x_i + \sum_{i=1}^{n-2} d_i x_i x_{i+1} = \\ &= 2 \sum_{i=0}^{n-2} d_i x_{i+1}^2 + 2 \sum_{i=1}^{n-1} d_i x_i^2 + 2 \sum_{i=1}^{n-2} d_i x_{i+1} x_i > 0, \end{aligned}$$

deoarece $d_i = t_{i+1} - t_i > 0, i = \overline{0, n-1}$. Rezultă că din sistemul (1.8) coeficienții a_1, a_2, \dots, a_{n-1} se determină în mod unic. Prin urmare funcția spline interpolatoare $g(t)$ este de asemenea unic determinată după formulele (1.6).

În cele prezentate mai sus ne-am limitat la analiza funcțiilor spline cubice ce satisfac condițiile la limită (1.2) de „ațîrnare liberă” a curbei interpolatoare în punctele t_0 și t_n . În practică însă adesea sînt cunoscute pantele curbei interpolatoare în punctele de frontieră. În acest caz se impune folosirea condițiilor la limită:

$$g'(t_0) = f_0', \quad g'(t_n) = f_n'. \quad (1.9)$$

Condițiile (1.9) cuplate cu (1.7) conduc la egalitățile:

$$2d_0 a_0 + d_0 a_1 = \frac{f_1 - f_0}{d_0} - f_0',$$

$$d_{n-1} a_{n-1} + 2d_{n-1} a_n = f_n' - \frac{f_n - f_{n-1}}{d_{n-1}}.$$

Aceste egalități se adaugă la ecuațiile (1.8), rezultînd un sistem liniar de $n+1$ ecuații cu $n+1$ necunoscute $a_0, a_1, a_2, \dots, a_n$, care are de asemenea soluție unică.

În alte cazuri practice este cunoscută curbura în punctele de ațîrnare t_0 și t_n :

$$g''(t_0) = f_0'', \quad g''(t_n) = f_n''. \quad (1.10)$$

În această situație sistemul liniar (1.8) se completează cu egalitățile:

$$a_0 = \frac{f_0''}{6}, \quad a_n = \frac{f_n''}{6}$$

care nu modifică matricea sistemului, ci afectează numai termenii liberi din prima și ultima ecuație.

În sfârșit, dacă se știe a priori că funcția $f(t)$ este periodică de perioadă $t_n - t_0$, atunci se impune aceeași condiție funcției interpolatoare $g(t)$, rezultând condițiile la limită:

$$g'(t_0) = g'(t_n), \quad g''(t_0) = g''(t_n). \quad (1.11)$$

În această situație avem evident $f_0 = f_n$. Din condițiile de periodicitate (1.11) rezultă alte două ecuații care trebuie adăugate sistemului (1.8):

$$a_0 = a_n,$$

$$2(d_0 + d_{n-1})a_0 + d_0a_1 + d_{n-1}a_{n-1} = \frac{f_1 - f_0}{d_0} - \frac{f_0 - f_{n-1}}{d_{n-1}}.$$

4.2. Interpretarea fizică a interpolării cu funcții spline

Funcțiile spline cubice au o proprietate extremală foarte importantă care asigură o înaltă efectivitate pentru acest tip de interpolare. Să considerăm clasa $W_{[t_0, t_n]}^{(2)}$ a funcțiilor care au derivate de ordinul doi cu pătrat integrabil pe segmentul $[t_0, t_n]$. Punem problema găsirii funcției interpolative:

$$u \in W_{[t_0, t_n]}^{(2)}, \quad u(t_i) = f_i, \quad i = \overline{0, n} \quad (2.1)$$

care minimizează funcționala:

$$\Phi(u) = \int_{t_0}^{t_n} [u''(t)]^2 dt \quad (2.2)$$

pe clasa de funcții $W_{[t_0, t_n]}^{(2)}$. Vom demonstra că minimul acestei funcționale este atins pe funcția spline $g(t)$ cubică pe porțiuni pe care am construit-o în secțiunea 4.1. În acest scop considerăm mărimea:

$$\Phi(u - g) = \int_{t_0}^{t_n} [u''(t) - g''(t)]^2 dt \quad (2.3)$$

unde $u(t)$ este o funcție interpolatoare oarecare care îndeplinește condițiile (2.1), iar $g(t)$ este funcția spline construită în secțiunea 4.1. Integrând prin părți în (2.3) și folosind proprietățile funcțiilor $g(t)$ și $u(t)$, obținem:

$$\begin{aligned}
 \Phi(u - g) &= \int_{t_0}^{t_n} [u''(t)]^2 dt - \int_{t_0}^{t_n} [g''(t)]^2 dt - 2 \int_{t_0}^{t_n} [u''(t) - g''(t)] g''(t) dt = \\
 &= \Phi(u) - \Phi(g) - 2 \left[(u' - g') g'' \Big|_{t=t_0}^{t=t_n} - \int_{t_0}^{t_n} [u'(t) - g'(t)] g'''(t) dt \right] = \\
 &= \Phi(u) - \Phi(g) + 2 \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} [u'(t) - g'(t)] g'''(t) dt.
 \end{aligned}$$

Am folosit condițiile la limită $g''(t_0) = g''(t_n) = 0$. Deoarece $g'''(t) = c_i = \text{const.}$ pe segmentul $[t_i, t_{i+1}]$, rezultă:

$$\begin{aligned}
 \Phi(u - g) &= \Phi(u) - \Phi(g) + 2 \sum_{i=0}^{n-1} c_i \left\{ [u(t_{i+1}) - g(t_{i+1})] - [u(t_i) - g(t_i)] \right\} = \\
 &= \Phi(u) - \Phi(g).
 \end{aligned}$$

Deoarece (2.3) implică evident $\Phi(u - g) \geq 0$ pentru orice u din clasa $W_{[t_0, t_n]}^{(2)}$, obținem:

$$\Phi(g) \leq \Phi(u) \quad (2.4)$$

pentru orice funcție u care îndeplinește condițiile (2.1). Deci minimul funcționalei (2.2) se realizează pe funcția spline cubică pe porțiuni $g(t)$, și nu este greu de arătat că alte puncte de minim ale funcționalei nu există.

Se poate demonstra că funcțiile spline cubice construite cu alte condiții la limită decât (1.2), de exemplu (1.9), (1.10) sau (1.11), asigură în continuare minimizarea funcționalei (2.2), însă acum nu pe întreaga clasă de funcții din (2.1), ci pe o submulțime a acesteia, alcătuită din funcții care satisfac condițiile la limită respective.

Pe baza rezultatului (2.4) se poate formula o alta definiție, echivalentă, a funcției spline cubice pe porțiuni: este acea funcție din clasa $W_{[t_0, t_n]}^{(2)}$ care ia valori date în nodurile rețelei și minimizează funcționala (2.2). Această proprietate a funcției spline este importantă prin faptul că funcționala $\Phi(u)$ poate fi interpretată ca analogul energiei potențiale a unei bare elastice forțată să treacă prin punctele (t_i, f_i) ale planului, pe funcțiile spline cubice realizându-se minimul acestei energii, respectiv „ondularea” minimă a barei elastice. Aceasta este proprietatea extremală „naturală” a funcțiilor spline cubice, care face ca acest tip de interpolare să fie cel mai potrivit pentru modelarea curbilor de evoluție a diverselor fenomene reale.

4.3. Considerații de implementare

În cazul abordării practice a interpolării curbilor plane pentru

dispozitivele periferice cu afișare grafică din sistemele de calcul, problema se pune în modul următor: se dă o mulțime finită de puncte din plan $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, ale căror coordonate x_i și y_i pot fi considerate numere întregi, deoarece în general spațiul de afișare grafică este discret. Se cere să se genereze curba interpolatoare care trece prin punctele respective și care are caracteristici de netezime cât mai bune. În cazul unui display grafic cu rastru rectangular, aceasta revine la a selecta pe ecran punctele (pixelii) care aproximează cel mai bine curba interpolatoare teoretică, ideală. În cazul unui plotter, problema se rezolvă prin generarea unui număr suficient de mare de puncte intermediare pe curba interpolatoare ideală, astfel încât între oricare două puncte consecutive curba să poată fi aproximată printr-o linie dreaptă.

În general, nici unul din șirurile x_0, x_1, \dots, x_n și y_0, y_1, \dots, y_n nu este strict crescător sau strict descrescător, prin urmare dezvoltarea teoretică prezentată în secțiunea 4.1 nu poate fi aplicată direct. Se impune abordarea parametrică a problemei, adică selectarea unui șir de valori $t_0 < t_1 < t_2 < \dots < t_n$ ale unei variabile independente continue t , și rezolvarea a două probleme teoretice de interpolare, una pentru șirul absciselor $(t_0, x_0), (t_1, x_1), \dots, (t_n, x_n)$ și alta pentru șirul ordonatelor $(t_0, y_0), (t_1, y_1), \dots, (t_n, y_n)$. Se obțin două funcții interpolatoare $x(t)$ și $y(t)$, $t \in [t_0, t_n]$ astfel încât $x(t_i) = x_i$, $y(t_i) = y_i$, $i = \overline{0, n}$, care definesc parametric curba care trebuie efectiv trasată.

Abordarea parametrică a problemei interpolării prezintă următorul avantaj: la o alegere inițială judicioasă a șirului de valori t_0, t_1, \dots, t_n ale variabilei independente, sistemul liniar (1.8) capătă o formă simplificată, care facilitează rezolvarea lui teoretică. Concret, vom considera $t_0 = 0, t_1 = 1, \dots, t_n = n$, ceea ce nu afectează generalitatea problemei. În acest caz $d_i = 1, \forall i$ și sistemul (1.8) devine:

$$a_{i-1} + 4a_i + a_{i+1} = f_{i-1} - 2f_i + f_{i+1}, \quad i = \overline{1, n-1} \quad (3.1)$$

$$a_0 = a_n = 0$$

unde $f_i = x_i$, respectiv $f_i = y_i$, $i = \overline{0, n}$ pentru cele două probleme de interpolare.

Programul care implementează interpolarea curbilor plane prin metoda funcțiilor spline cubice, trebuie să realizeze două sarcini majore și principal distincte:

- Să rezolve două sisteme liniare de tipul (3.1), separat pentru șirul absciselor $\{x_i\}$, respectiv al ordonatelor $\{y_i\}$, $i = \overline{0, n}$, obținând două seturi de câte $n+1$ parametri $\{a_i\}$, $i = \overline{0, n}$ care definesc funcțiile interpolatoare cubice pe porțiuni $x(t)$, respectiv $y(t)$, conform formulelor de tipul (1.6);

- Pe fiecare interval $[t_i, t_{i+1}] = [i, i+1]$, să genereze efectiv arcul de curbă definită parametric prin $x = x(t), y = y(t)$, care unește două puncte consecutive date (x_i, y_i) și (x_{i+1}, y_{i+1}) , unde $x(t)$ și $y(t)$ sînt definite prin formule de tipul (1.6).

A doua parte a programului poate fi considerată ca o rutină independentă de generare a imaginii discretizate a unei curbe continue plane, definită parametric prin două funcții polinomiale de gradul 3. Această rutină va fi discutată în detaliu în secțiunile 4.7 și 4.8.

În secțiunea următoare va fi prezentată o metodă de rezolvare a sistemelor liniare de tipul (3.1), deducîndu-se formule de calcul exact sau aproximativ al parametrilor curbei interpolatoare de tip spline.

4.4. Determinarea analitică a parametrilor curbei

Aceasta revine la rezolvarea sistemului liniar de $n-1$ ecuații cu $n-1$ necunoscute $a_i, i = \overline{1, n-1}$:

$$\begin{aligned} a_{i-1} + 4a_i + a_{i+1} &= h_i, \quad i = \overline{1, n-1} \\ a_0 &= a_n = 0, \end{aligned} \quad (4.1)$$

unde am notat $h_i = f_{i-1} - 2f_i + f_{i+1}$, $i = \overline{1, n-1}$, iar f_i reprezintă de fapt valori de abscise sau ordinate grafice x_i sau y_i . După cum s-a arătat în secțiunea 4.3, f_i și prin urmare h_i pot fi considerate numere întregi.

Vom folosi condițiile la limită simple (1.2). Pentru alte condiții la limită, de exemplu (1.9), (1.10) sau (1.11), sistemul se rezolvă în mod asemănător, cu o complicație neesențială a calculelor.

Introducem constanta:

$$\alpha = -2 - \sqrt{3}, \quad \alpha + \frac{1}{\alpha} = -4 \quad (4.2)$$

care este rădăcina de modul > 1 a ecuației $\alpha^2 + 4\alpha + 1 = 0$. Putem rescrie sistemul (4.1) astfel:

$$\frac{a_{i-1} - \alpha a_i}{\alpha^i} - \frac{a_i - \alpha a_{i+1}}{\alpha^{i+1}} = \frac{h_i}{\alpha^i}, \quad i = \overline{1, n-1}.$$

Adunînd primele $i-1$ ecuații ale sistemului, obținem:

$$\frac{a_0 - \alpha a_1}{\alpha} - \frac{a_{i-1} - \alpha a_i}{\alpha^i} = \sum_{k=1}^{i-1} \frac{h_k}{\alpha^k}, \quad i = \overline{2, n}.$$

Deoarece $a_0 = 0$, rezultă:

$$\alpha^i a_i - \alpha^{i-1} a_{i-1} = \alpha^{2i-1} a_1 + \sum_{k=1}^{i-1} \alpha^{2i-1-k} h_k, \quad i = \overline{2, n}.$$

Adunând din nou primele $i-1$ ecuații ale sistemului de mai sus, obținem:

$$\alpha^i a_i - \alpha a_1 = \alpha a_1 \sum_{j=2}^i \alpha^{2j-2} + \sum_{j=2}^i \sum_{k=1}^{j-1} \alpha^{2j-1-k} h_k, \quad i = \overline{2, n}.$$

Prin urmare:

$$\begin{aligned} \alpha^i a_i &= \alpha a_1 \sum_{j=1}^i \alpha^{2(j-1)} + \sum_{j=2}^i \sum_{k=1}^{j-1} \alpha^{2j-1-k} h_k = \\ &= \frac{\alpha(\alpha^{2i} - 1)}{\alpha^2 - 1} a_1 + \sum_{j=1}^{i-1} \sum_{k=1}^j \alpha^{2j+1-k} h_k, \quad i = \overline{2, n}. \end{aligned}$$

Folosind identitatea elementară $\sum_{j=1}^i \sum_{k=1}^j p_{jk} q_k = \sum_{k=1}^i q_k \sum_{j=k}^i p_{jk}$, deducem:

$$\begin{aligned} \alpha^i a_i &= \frac{\alpha(\alpha^{2i} - 1)}{\alpha^2 - 1} a_1 + \sum_{k=1}^{i-1} h_k \sum_{j=k}^{i-1} \alpha^{2j+1-k} = \\ &= \frac{\alpha(\alpha^{2i} - 1)}{\alpha^2 - 1} a_1 + \sum_{k=1}^{i-1} \alpha^{2k+1-k} h_k \sum_{j=0}^{i-k-1} \alpha^{2j} = \\ &= \frac{\alpha(\alpha^{2i} - 1)}{\alpha^2 - 1} a_1 + \sum_{k=1}^{i-1} \frac{\alpha^{k+1}(\alpha^{2i-2k} - 1)}{\alpha^2 - 1} h_k = \\ &= \frac{\alpha(\alpha^{2i} - 1)}{\alpha^2 - 1} a_1 + \sum_{k=1}^{i-1} \frac{\alpha^{2i-k+1} - \alpha^{k+1}}{\alpha^2 - 1} h_k, \quad i = \overline{2, n}. \end{aligned}$$

Rezultă:

$$a_i = \frac{\alpha^i - \alpha^{-i}}{\alpha - \alpha^{-1}} a_1 + \sum_{k=1}^{i-1} \frac{\alpha^{i-k} - \alpha^{k-i}}{\alpha - \alpha^{-1}} h_k, \quad i = \overline{2, n}. \quad (4.3)$$

Folosind condiția $a_n = 0$, deducem valoarea lui a_1 :

$$a_1 = \frac{1}{\alpha^n - \alpha^{-n}} \sum_{k=1}^{n-1} (\alpha^{k-n} - \alpha^{n-k}) h_k. \quad (4.4)$$

Revenind la (4.3), obținem pentru $i = \overline{2, n-1}$:

$$\begin{aligned}
 a_i &= \frac{\alpha^i - \alpha^{-i}}{(\alpha - \alpha^{-1})(\alpha^n - \alpha^{-n})} \sum_{k=1}^{n-1} (\alpha^{k-n} - \alpha^{n-k}) h_k + \frac{1}{\alpha - \alpha^{-1}} \sum_{k=1}^{i-1} (\alpha^{i-k} - \alpha^{k-i}) h_k = \\
 &= \frac{1}{(\alpha - \alpha^{-1})(\alpha^n - \alpha^{-n})} \left\{ \sum_{k=1}^{i-1} [(\alpha^i - \alpha^{-i})(\alpha^{k-n} - \alpha^{n-k}) + (\alpha^n - \alpha^{-n})(\alpha^{i-k} - \alpha^{k-i})] h_k + \right. \\
 &\quad \left. + \sum_{k=i}^{n-1} (\alpha^i - \alpha^{-i})(\alpha^{k-n} - \alpha^{n-k}) h_k \right\} = \\
 &= \frac{1}{(\alpha - \alpha^{-1})(\alpha^n - \alpha^{-n})} \left[\sum_{k=1}^{i-1} (\alpha^{n-i-k} + \alpha^{-n+i+k} - \alpha^{n-i+k} - \alpha^{-n+i-k}) h_k + \right. \\
 &\quad \left. + \sum_{k=i}^{n-1} (\alpha^{n-i-k} + \alpha^{-n+i+k} - \alpha^{n+i-k} - \alpha^{-n-i+k}) h_k \right]. \quad (4.5)
 \end{aligned}$$

Alte exprimări echivalente pentru expresiile soluțiilor $a_i, i = \overline{2, n-1}$ sînt următoarele:

$$\begin{aligned}
 a_i &= \frac{1}{(\alpha - \alpha^{-1})(\alpha^n - \alpha^{-n})} \left[(\alpha^{n-i} - \alpha^{i-n}) \sum_{k=1}^{i-1} (\alpha^{-k} - \alpha^k) h_k + \right. \\
 &\quad \left. + (\alpha^i - \alpha^{-i}) \sum_{k=i}^{n-1} (\alpha^{k-n} - \alpha^{n-k}) h_k \right] \quad (4.6)
 \end{aligned}$$

$$\begin{aligned}
 a_i &= \frac{1}{(\alpha - \alpha^{-1})(\alpha^n - \alpha^{-n})} \sum_{k=1}^{n-1} (\alpha^{n-i-k} + \alpha^{-n+i+k} - \alpha^{n-|i-k|} - \alpha^{-n+|i-k|}) h_k, \\
 i &= \overline{0, n}. \quad (4.7)
 \end{aligned}$$

Ultima exprimare (4.7) este general valabilă pentru toate soluțiile $a_i, i = \overline{0, n}$.

Definim șirurile de numere:

$$A_n = \frac{\alpha^n + \alpha^{-n}}{2}, \quad B_n = \frac{\alpha^n - \alpha^{-n}}{\alpha - \alpha^{-1}}, \quad n \geq 0. \quad (4.8)$$

Numele A_n, B_n sînt toate întregi și respectă aceeași relație de recurență:

$$A_{n+1} + 4A_n + A_{n-1} = 0, \quad B_{n+1} + 4B_n + B_{n-1} = 0, \quad n \geq 1. \quad (4.9)$$

Primele 10 valori ale celor două șiruri sînt listate în tabelul următor.

| | | | | | | | | | | |
|---------|---|----|----|-----|-----|------|------|-------|--------|--------|
| $n =$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| $A_n =$ | 1 | -2 | 7 | -26 | 97 | -362 | 1351 | -5042 | 18817 | -70226 |
| $B_n =$ | 0 | 1 | -4 | 15 | -56 | 209 | -780 | 2911 | -10864 | 40545 |

Cu aceasta, expresiile parametrilor de interpolare (4.4) și (4.7) se rescriu după cum urmează:

$$a_1 = -\frac{1}{B_n} \sum_{k=1}^{n-1} B_{n-k} h_k \quad (4.10)$$

$$a_i = \frac{1}{6B_n} \sum_{k=1}^{n-1} (A_{|n-i-k|} - A_{n-|i-k|}) h_k, \quad i = \overline{0, n}. \quad (4.11)$$

Toate numerele care apar în expresiile din (4.10) și (4.11) sînt întregi, deci a_i sînt numere raționale, combinații liniare de elemente h_k , ceea ce rezultă direct din însăși structura sistemului liniar inițial. După cum era de așteptat, în expresia lui a_i elementul h_i are ponderea cea mai mare în valoare absolută ($A_n - A_{|n-2i|}$).

4.5. Calculul numeric al parametrilor curbei

Să analizăm acum posibilitățile de calculare numerică a parametrilor a_i . Deoarece numerele A_n și B_n cresc exponențial cu n , metodele exacte nu dau rezultate (produc depășiri aritmetice) pentru un număr suficient de mare de puncte de interpolare; trebuie găsite metode aproximative.

O primă idee „simplificatoare” ar fi să calculăm la început a_1 , a cărei expresie (4.10) este mai simplă, și apoi să deducem ceilalți coeficienți a_i pe baza relației de recurență inițiale (4.1). Aceasta este bineînțeles metoda cea mai rapidă deoarece nu necesită înmulțiri și împărțiri decât pentru calcularea lui a_1 (înmulțirile cu 4 care apar în relația de recurență pot fi executate prin deplasări binare la stînga). Totuși, metoda este numeric instabilă și nu este indicată pentru un număr mare de puncte de interpolare. Dacă, de exemplu, am calculat a_1 cu eroarea ε , pentru $a_2 = h_1 - 4a_1$ vom obține eroarea -4ε , pentru $a_3 = h_2 - 4a_2 - a_1$ eroarea 15ε , ... pentru a_n eroarea $B_n\varepsilon$, conform cu (4.9). Deci, în acest caz, eroarea crește exponențial cu numărul de puncte de interpolare.

Prin urmare, pentru a dezvolta o procedură de calcul aproximativ al parametrilor a_i , trebuie să pornim de la egalitatea (4.11).

Pentru un număr redus de puncte de interpolare, de exemplu $n \leq 8$,

expresia din (4.11) poate fi calculată exact (mai puțin ultima împărțire) folosind numai operații elementare în virgulă fixă; într-adevăr, $|6B_n| \leq 2^{16}$ și $|A_{|n-i-k|}| < |A_{n-|i-k|}| \leq |A_n| < 2^{15}$ sînt cantități care pot fi reprezentate pe 16 biți (am folosit faptul că $|n-i-k| < n-|i-k|$ pentru $i, k = \overline{1, n-1}$, precum și faptul că șirul $|A_n|$ este strict crescător). De asemenea, $h_k = f_{k-1} - 2f_k + f_{k+1}$ sînt numere întregi cel mult de ordinul miilor, deoarece f_k sînt coordonate horizontale sau verticale în spațiul grafic de afișare. Prin urmare, pentru $n \leq 8 \cdot a_i$ se poate calcula sub forma unui raport de numere întregi $\frac{P_i}{Q_i}$, unde P_i este un număr de 32 biți, iar Q_i de 16 biți.

Pentru $n \geq 9$, dorim să obținem evaluări asemănătoare, aproximative, pentru parametrii a_i . Este necesar să găsim aproximări bune pentru numerele raționale:

$$\frac{A_{n-|i-k|} - A_{|n-i-k|}}{6B_n} = \frac{A_{p+2q} - A_p}{6B_n}$$

unde am notat $p = |n-i-k|$, $p+2q = n-|i-k|$ și am folosit faptul că indicii $n-|i-k|$ și $|n-i-k|$ au aceeași paritate.

Vom folosi aproximări de tipul:

$$\frac{A_p}{B_{l+m}} \approx \begin{cases} 0 & \text{pentru } p < m \\ \frac{A_{p-m}}{B_l} & \text{pentru } p \geq m \end{cases} \quad (5.1)$$

unde $l \leq 9$, astfel încît B_l să poată fi reprezentat pe 16 biți.

Aceste expresii se bazează pe ideea:

$$\frac{\alpha^p + \alpha^{-p}}{\alpha^{l+m} - \alpha^{-l-m}} \approx \frac{\alpha^p}{\alpha^{l+m}} = \frac{\alpha^{p-m}}{\alpha^l} \approx \frac{\alpha^{p-m} + \alpha^{-p+m}}{\alpha^l - \alpha^{-l}}.$$

Prin calcul direct se dovedește că aceasta este cea mai bună aproximare posibilă de tipul $\frac{\alpha^p + \alpha^{-p}}{\alpha^{l+m} - \alpha^{-l-m}} \approx \frac{\alpha^{p-m} \pm \alpha^{-p+m}}{\alpha^l \pm \alpha^{-l}}$. Cu cît l este mai mare, cu atît aproximarea este mai bună.

Prin urmare, vom lua $l = 9$ și vom nota $m = n - l = n - 9$. În conformitate cu (5.1), deducem:

$$\frac{A_{p+2q} - A_p}{6B_n} \approx \begin{cases} 0 & \text{pentru } p+2q < m \\ \frac{A_{p+2q-m}}{6B_9} & \text{pentru } p < m \leq p+2q \\ \frac{A_{p+2q-m} - A_{p-m}}{6B_9} & \text{pentru } p \geq m. \end{cases} \quad (5.2)$$

Să observăm acum că direct din relațiile de definiție ale lui A_n și B_n (4.8) rezultă:

$$A_{2n} = 6B_n^2 + 1, A_{2n+1} = 6B_nB_{n+1} - 2, n \geq 0.$$

Definim șirul de numere întregi $\{C_n\}_{n \geq 0}$ astfel:

$$\begin{cases} C_{2n} = B_n^2 \\ C_{2n+1} = B_nB_{n+1}. \end{cases} \quad (5.3)$$

Numărul C_n este cea mai bună aproximare întreagă a lui $\frac{A_n}{6}$. Pe de altă parte, are loc egalitatea *exactă*:

$$\frac{A_{n+2k} - A_n}{6} = C_{n+2k} - C_n, n, k \geq 0.$$

Primele 10 valori ale șirului C_n sînt următoarele:

| | | | | | | | | | | |
|---------|---|---|---|----|----|-----|-----|------|------|--------|
| $n =$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| $C_n =$ | 0 | 0 | 1 | -4 | 16 | -60 | 225 | -840 | 3136 | -11704 |

Folosind numerele C_n , putem rescrie evaluarea aproximativă (5.2) astfel:

$$\frac{A_{p+2q} - A_p}{6B_n} \approx \begin{cases} 0 & \text{pentru } p+2q-1 \leq m \\ \frac{C_{p+2q-m}}{B_9} & \text{pentru } p-1 \leq m \leq p+2q-2 \\ \frac{C_{p+2q-m} - C_{p-m}}{B_9} & \text{pentru } p-2 \geq m. \end{cases} \quad (5.4)$$

Am folosit în plus $C_0 = C_1 = 0$.

Pe baza aproximărilor (5.4), pornind de la (4.11), expresia parametrului a_i devine:

$$\begin{aligned} a_i &= -\frac{1}{6B_n} \left[\sum_{k=1}^{j-1} (A_{n-i+k} - A_{|n-i-k|}) h_k + \sum_{k=i}^{n-1} (A_{n+i-k} - A_{|n-i-k|}) h_k \right] \approx \\ &\approx -\frac{1}{B_9} \left[\sum_{k=1}^{j-1} (C_{n-i+k-m} - C_{|n-i-k|-m}) h_k + \right. \\ &\quad \left. + \sum_{k=i}^{n-1} (C_{n+i-k-m} - C_{|n-i-k|-m}) h_k \right] \end{aligned} \quad (5.5)$$

dacă introducem convenția suplimentară $C_n = 0$ pentru $n < 0$ (mai exact

$C_n = 0$ pentru $n < 2$).

Procedura concretă de calcul al lui a_i , pentru n și i date, trebuie să adune toți termenii nenuli din sumele din (5.5), care se centrează în jurul termenului de pondere maximă, de indice $k = i$. Se observă din (5.5) că ponderile elementelor h_k descresc pe măsură ce indicele k se depărtează de i , crescător sau descrescător.

Indicele k minim care dă un termen nenul rezultă din egalitatea $C_{n-i+k-m} = C_2$, deci $k_{\min} = -n+i+m+2 = i-7$. Bineînțeles, dacă $i \leq 7$ trebuie considerat $k_{\min} = 1$.

Indicele k maxim care dă un termen nenul rezultă din egalitatea $C_{n+i-k-m} = C_2$, deci $k_{\max} = n+i-m-2 = i+7$. Bineînțeles, dacă $i \geq n-7$ trebuie considerat $k_{\max} = n-1$.

Prezentăm în continuare o procedură de calcul al parametrului a_i . Procedura folosește 8 constante întregi C_2, C_3, \dots, C_9 , prezentate mai sus, care pot fi stocate pe 16 biți fiecare, într-o tabelă din memorie. Tot pe 16 biți se memorează constanta $B_9 = 40545$. După cum am arătat mai înainte, h_k sînt numere întregi care de asemenea pot fi reprezentate pe 16 biți. Pentru evitarea riscului de depășire aritmetică, rezultatele înmulțirilor și sumele parțiale se calculează pe 32 biți. Procedura va genera numărul a_i sub forma $P_i/40545$, unde P_i este un întreg de 32 biți.

Procedură de calcul pentru parametrul a_i

Intrări: $n \geq 9, i, h_k = f_{k-1} - 2f_k + f_{k+1}, k = \overline{1, n-1}$.

leșiri: P_i, a_i .

Pasul 1: Inițializează $m \leftarrow n-9, k \leftarrow i-7, k_{\max} \leftarrow i+7, P_i \leftarrow 0$;
dacă $k \leq 0$, atunci $k \leftarrow 1$;
dacă $k_{\max} \geq n$, atunci $k_{\max} \leftarrow n-1$.

Pasul 2: Dacă $k = i$, salt la Pasul 6.

Pasul 3: Calculează $j_1 \leftarrow 9-i+k, j_2 \leftarrow |n-i-k|-m$.

Pasul 4: Dacă $j_2 \geq 2$, calculează $P_i \leftarrow P_i + (C_{j_1} - C_{j_2}) h_k$;
dacă $j_2 < 2$, calculează $P_i \leftarrow P_i + C_{j_1} h_k$.

Pasul 5: Calculează $k \leftarrow k+1$ și salt la Pasul 2.

Pasul 6: Calculează $j_1 \leftarrow 9+i-k, j_2 \leftarrow |n-i-k|-m$.

Pasul 7: Dacă $j_2 \geq 2$, calculează $P_i \leftarrow P_i + (C_{j_1} - C_{j_2}) h_k$;
dacă $j_2 < 2$, calculează $P_i \leftarrow P_i + C_{j_1} h_k$.

Pasul 8: Calculează $k \leftarrow k+1$.

Pasul 9: Dacă $k \leq k_{\max}$ salt la Pasul 6.

Pasul 10: Calculează $a_i \leftarrow -\frac{P_i}{40545}$ și stop.

De exemplu, pentru $n = 9$ și $i = 1$ procedura generează:

$$a_1 = \frac{10864 h_1 - 2911 h_2 + 780 h_3 - 209 h_4 + 56 h_5 - 15 h_6 + 4 h_7 - h_8}{40545}$$

care este de fapt expresia *exactă* a lui a_1 , exceptînd ultima împărțire (a se compara cu (4.10)).

Procedura prezentată mai sus ia în considerare valorile funcției în maximum 15 puncte de interpolare în jurul punctului central de indice i , și anume punctele de indici $i-7, i-6, \dots, i, i+1, \dots, i+7$. Valorile funcției în celelalte puncte de interpolare sînt neglijate. Aceasta este suficient pentru a asigura o înaltă precizie interpolării bazate pe metoda de calcul expusă mai sus. Precizia rezultatului final al interpolării este limitată în primul rînd de structura discretă (rezoluția finită) a mediului de afișare, și nu de aproximările aritmetice, chiar dacă se folosesc numai operații în virgulă fixă.

Practic, pentru mediile de afișare care se întîlnesc uzual, este probabil suficient să se ia în considerare valorile funcției în 7 sau 9 puncte de interpolare în jurul punctului de indice i . În acest scop, procedura de mai sus pentru calculul parametrului a_i se poate generaliza foarte simplu; este suficient să se considere o valoare $l < 9$ în formula (5.1). Rezultă pentru $m = n-l$ o valoare mai apropiată de n , iar pentru constanta de la numitor se folosește B_l . Numărul K_l de puncte de interpolare luate în considerare pentru generarea unui arc de curbă se reduce la $K_l = 2l-3 < 15$.

Timpul de execuție al procedurii este aproximativ proporțional cu l . Totuși, cîștigul de timp obținut reducînd pe l , respectiv pe K_l , este nesemnificativ în raport cu timpul total al operației de interpolare; acesta este determinat în primul rînd de eficiența procedurii care desenează efectiv un arc de curbă între două puncte de interpolare consecutive. Această procedură, care va fi discutată în secțiunile 4.7 și 4.8, este evident independentă de K_l .

4.6. Interpolare cu un număr redus de puncte

Rezultatele prezentate în secțiunile anterioare au pornit de la ipoteza implicită că toate coordonatele punctelor fixe $x_i, y_i, i = \overline{0, n}$ sînt

cunoscute înainte de trasarea curbei interpolatoare. În unele situații practice, întâlnite mai ales în aplicațiile grafice interactive, această condiție inițială nu este îndeplinită: interpolarea propriu-zisă trebuie să înceapă înainte de specificarea tuturor punctelor. De exemplu, interpolatorul ReGIS, implementat pe terminalele grafice ale firmei DIGITAL, pornește interpolarea după specificarea primelor 4 puncte P_0, P_1, P_2 și P_3 : în momentul când se specifică P_3 , se trasează curba de interpolare între P_1 și P_2 . În continuare, pentru fiecare punct P_i specificat, $i = 4, 5, 6, \dots$, se generează un arc de curbă interpolatoare între cele două puncte anterioare P_{i-2} și P_{i-1} .

În această secțiune vom analiza problema de interpolare de mare interes practic descrisă mai sus, vom deduce formule de calcul pentru parametrii curbei interpolatoare între două puncte consecutive P_i și P_{i+1} , și vom prezenta o procedură de principiu pentru trasarea curbei interpolatoare.

Deoarece dorim să rezolvăm problema tot cu funcții interpolatoare cubice pe porțiuni, vom relua formula (1.3) care definește derivata a doua a funcției $g(t)$ pe intervalul $[t_i, t_{i+1}]$. Ținând seamă de simplificarea $t_i = i, d_i = 1$ pentru $i = 0, 1, 2, \dots$, discutată în secțiunea 4.3, putem scrie:

$$g''(t) = 6a_i(i+1-t) + 6b_i(t-i), \quad t \in [i, i+1] \quad (6.1)$$

unde $6a_i = g''(i+0)$, $6b_i = g''(i+1-0)$.

Să facem abstracție pentru moment de ipoteza continuității derivatei a doua. Procedînd la fel ca în secțiunea 4.1, deducem succesiv:

$$g(t) = a_i(i+1-t)^3 + b_i(t-i)^3 + (f_i - a_i)(i+1-t) + (f_{i+1} - b_i)(t-i) \quad (6.2)$$

$$g'(t) = -3a_i(i+1-t)^2 + 3b_i(t-i)^2 + (f_{i+1} - f_i) - (b_i - a_i)$$

$$g'(i+0) = -2a_i - b_i + (f_{i+1} - f_i) \quad (6.3)$$

$$g'(i+1-0) = 2b_i + a_i + (f_{i+1} - f_i)$$

$$g'(i-0) = 2b_{i-1} + a_{i-1} + (f_i - f_{i-1}).$$

Specificul problemei impune aflarea parametrilor a_i și b_i din (6.2), necesari pentru generarea curbei între punctele i și $i+1$, în funcție de valorile cunoscute $f_0, f_1, \dots, f_i, f_{i+1}, f_{i+2}$. De asemenea, parametrii a_{i-1}, b_{i-1} care determină funcția $g(t)$ între $i-1$ și i trebuie considerați cunoscuți, deoarece curba este deja trasată între P_{i-1} și P_i , când se specifică punctul P_{i+2} . Exprimînd condițiile de continuitate ale derivatei $g'(t)$ în punctele $i, i+1, i+2, \dots$ (numărul punctelor este deocamdată neprecizat), din (6.3) deducem:

$$a_{i-1} + 2b_{i-1} + 2a_i + b_i = h_i \quad (6.4)$$

$$a_i + 2b_i + 2a_{i+1} + b_{i+1} = h_{i+1}$$

$$a_{i+1} + 2b_{i+1} + 2a_{i+2} + b_{i+2} = h_{i+2}$$

... ..

unde am notat ca mai înainte $h_i = f_{i-1} - 2f_i + f_{i+1}$.

Dacă punem condiția de continuitate a derivatei a doua $g''(t)$ în punctul i , rezultă $a_i = b_{i-1}$, iar prima ecuație (6.4) permite determinarea lui $b_i = h_i - a_{i-1} - 4b_{i-1}$. Aceasta înseamnă că parametrii curbei interpolatoare între punctele P_i și P_{i+1} se calculează fără să se țină seama de coordonatele punctului P_{i+2} , care sînt cunoscute conform datelor problemei. Acest aspect nu poate să nu degradeze calitatea interpolării, cu toate că din punct de vedere matematic curba va avea caracteristici de netezime superioare. Într-adevăr, arcul de curbă interpolatoare între P_i și P_{i+1} , depinzînd exclusiv de punctele $P_0, P_1, \dots, P_i, P_{i+1}$, nu „anticipează” traseul dincolo de P_{i+1} , deși ar putea s-o facă datorită faptului că punctul P_{i+2} este cunoscut.

Prin urmare, pentru $i > 1$, vom renunța la condiția de continuitate a lui $g''(t)$ în punctul i și vom păstra două necunoscute a_i și b_i . În schimb, putem adăuga condițiile de continuitate ale derivatei a doua $g''(t)$ în punctele $i+1, i+2$ și în eventualele puncte ulterioare $i+3, i+4, \dots$, de unde rezultă $a_{j+1} = b_j$ pentru $j \geq i$. Cu aceasta, relațiile (6.4) devin:

$$a_{i-1} + 2b_{i-1} + 2a_i + b_i = h_i \quad (6.5)$$

$$a_i + 4b_i + b_{i+1} = h_{i+1}$$

$$b_{j-1} + 4b_j + b_{j+1} = h_{j+1}, \quad j \geq i+1.$$

În ecuațiile (6.5), termenii liberi h_i și h_{i+1} sînt cunoscuți, dar h_{i+2}, h_{i+3}, \dots nu sînt cunoscuți, deoarece depind de coordonatele punctelor ulterioare P_{i+3}, P_{i+4}, \dots . Un mod grosier de a elimina această dificultate este de a ține seamă numai de primele două ecuații (6.5) considerînd în plus $b_{i+1} = 0$; aceasta revine la a determina parametrii a_i și b_i ai curbei între P_i și P_{i+1} în ipoteza de curbă nulă în punctul P_{i+2} .

Vom adopta o ipoteză mai realistă în această problemă, care constă în a presupune că urmează o infinitate de puncte de interpolare P_{i+3}, P_{i+4}, \dots plasate în continuarea liniei drepte care unește P_{i+1} și P_{i+2} , astfel încît toate punctele $P_{i+1}, P_{i+2}, P_{i+3}, \dots$ sînt echidistante pe această dreaptă. Prin urmare, vom considera o infinitate de ecuații în (6.5) luînd $h_{i+2} = h_{i+3} = \dots = 0$. De asemenea, în această situație ipotetică putem presupune că:

$$\lim_{j \rightarrow \infty} b_j = 0 \quad (6.6)$$

deoarece pe măsură ce numărul punctelor de interpolare crește la infinit, curba interpolatoare care trece prin ele tinde la linia dreaptă pe care acestea sînt plasate.

Aceste ipoteze plauzibile permit rezolvarea sistemului (6.5) după cum urmează. Mai întîi, vom rescrie a 3-a egalitate din (6.5) sub forma:

$$\frac{1}{\alpha^j}(b_{j-1} - \alpha b_j) - \frac{1}{\alpha^{j+1}}(b_j - \alpha b_{j+1}) = 0, \quad j \geq i+1 \quad (6.7)$$

unde am folosit constanta $\alpha = -2 - \sqrt{3}$ introdusă în (4.2). Adunînd primele p relații din (6.7), care corespund la $j = i+1, j = i+2, \dots, j = i+p$, obținem:

$$\frac{1}{\alpha^{i+1}}(b_i - \alpha b_{i+1}) - \frac{1}{\alpha^{i+p+1}}(b_{i+p} - \alpha b_{i+p+1}) = 0, \quad p \geq 1. \quad (6.8)$$

Făcînd $p \rightarrow \infty$ (i este fixat) în (6.8) și ținînd seama de (6.6), deducem $b_i - \alpha b_{i+1} = 0$ și în general $b_j - \alpha b_{j+1} = 0, j \geq i+1$. Înlocuim $b_{i+1} = \frac{1}{\alpha} b_i$ în primele două relații (6.5), pe care apoi le rescriem sub forma:

$$2a_i + b_i = h_i - a_{i-1} - 2b_{i-1} \quad (6.9)$$

$$a_i - \alpha b_i = h_{i+1}.$$

Sistemul rezultat (6.9) permite deducerea parametrilor a_i, b_i care determină pe $g(t)$ între punctele i și $i+1$, în funcție de parametrii a_{i-1} și b_{i-1} referitori la intervalul anterior $[i-1, i]$. Prin urmare, pentru primul arc de curbă care trebuie trasat între punctele P_1 și P_2 problema trebuie examinată separat.

Vom pune din nou condiția $g''(0) = 0$ de „afîrnare liberă” a curbei interpolatoare în punctul inițial $i = 0$, de unde rezultă $a_0 = 0$. În sfîrșit, putem pune condiția de continuitate a derivatei a doua $g''(t)$ în punctul $i = 1$, deoarece primul arc de curbă este generat fără restricții, prin urmare luăm $b_0 = a_1$.

În concluzie, parametrii $a_i, b_i, i \geq 1$ ai funcției interpolatoare $g(t)$ pe intervalul $[i, i+1]$ se determină din condițiile:

$$4a_1 + b_1 = h_1 \quad (6.10)$$

$$a_1 - \alpha b_1 = h_2$$

$$2a_i + b_i = h_i - a_{i-1} - 2b_{i-1}, \quad i \geq 2$$

$$a_i - \alpha b_i = h_{i+1}, \quad i \geq 2.$$

Din primele două ecuații (6.10) deducem parametrii a_1, b_1 pentru primul

arc de curbă, între P_1 și P_2 :

$$\begin{aligned} a_1 &= -\frac{h_1}{\alpha} - \frac{h_2}{\alpha^2} \\ b_1 &= \frac{-h_1 + 4h_2}{\alpha^2}. \end{aligned} \quad (6.11)$$

Pentru $i \geq 2$, din ultimele două ecuații (6.10) deducem succesiv:

$$\begin{aligned} a_i &= \frac{h_{i+1} + \alpha h_i - \alpha a_{i-1} - 2ab_{i-1}}{2\alpha + 1} = \frac{h_{i+1} + \alpha h_i - \alpha a_{i-1} - 2(a_{i-1} - h_i)}{2\alpha + 1} \\ b_i &= \frac{h_i - 2h_{i+1} - a_{i-1} - 2b_{i-1}}{2\alpha + 1} = \frac{h_i - 2h_{i+1} - (h_i + \alpha b_{i-1}) - 2b_{i-1}}{2\alpha + 1}. \end{aligned}$$

Am folosit faptul că egalitatea $a_i - \alpha b_i = h_{i+1}$ e valabilă și pentru $i = 1$. În definitiv, pentru $i \geq 2$ obținem relațiile de recurență simple:

$$\begin{aligned} a_i &= \frac{a_{i-1}}{\alpha} + \frac{h_{i+1} - \sqrt{3}h_i}{\sqrt{3}\alpha} \\ b_i &= \frac{b_{i-1}}{\alpha} - \frac{2h_{i+1}}{\sqrt{3}\alpha}. \end{aligned} \quad (6.12)$$

Datorită prezenței la numitor a factorului α de modul > 1 , relațiile (6.12) pot fi utilizate efectiv pentru calcularea iterativă a coeficienților a_i, b_i . O eventuală eroare de rotunjire în valoarea calculată a lui a_{i-1} sau b_{i-1} nu se amplifică, ci scade în progresie geometrică atunci când parametrii a_i, a_{i+1}, \dots și b_i, b_{i+1}, \dots se calculează aplicând iterativ relațiile (6.12).

Este interesant de estimat „nivelul de discontinuitate” a derivatei a doua $g''(t)$ a funcției interpolatoare, rezultat în urma procedurii de determinare a parametrilor a_i, b_i expus mai sus. Pentru aceasta, să evaluăm diferența $g''(i+0) - g''(i-0) = 6(a_i - b_{i-1})$ într-un punct oarecare i . Aplicând în mod recursiv formulele (6.12) și în final a doua formulă (6.10), deducem:

$$\begin{aligned} a_i &= \frac{a_{i-1}}{\alpha} + \frac{h_{i+1} - \sqrt{3}h_i}{\sqrt{3}\alpha} = \frac{a_1}{\alpha^{i-1}} + \sum_{j=2}^i \frac{h_{j+1} - \sqrt{3}h_j}{\sqrt{3}\alpha^{i-j+1}} = \\ &= \frac{a_1}{\alpha^{i-1}} + \sum_{j=3}^{i+1} \frac{h_j}{\sqrt{3}\alpha^{i-j+2}} - \sum_{j=2}^i \frac{h_j}{\alpha^{i-j+1}} \\ b_i &= \frac{b_{i-1}}{\alpha} - \frac{2h_{i+1}}{\sqrt{3}\alpha} = \frac{b_1}{\alpha^{i-1}} - \sum_{j=2}^i \frac{2h_{j+1}}{\sqrt{3}\alpha^{i-j+1}} = \frac{b_1}{\alpha^{i-1}} - \sum_{j=3}^{i+1} \frac{2h_j}{\sqrt{3}\alpha^{i-j+2}} \\ a_i - b_{i-1} &= \sum_{j=3}^{i+1} \frac{h_j}{\sqrt{3}\alpha^{i-j+2}} - \sum_{j=2}^i \frac{h_j}{\alpha^{i-j+1}} + \sum_{j=3}^i \frac{2h_j}{\sqrt{3}\alpha^{i-j+1}} + \frac{a_1}{\alpha^{i-1}} - \frac{b_1}{\alpha^{i-2}} = \end{aligned}$$

$$= \sum_{j=3}^i \frac{h_j}{\alpha^{j-j+1}} \left(\frac{1}{\sqrt{3}\alpha} - 1 + \frac{2}{\sqrt{3}} \right) + \frac{h_{i+1}}{\sqrt{3}\alpha} - \frac{h_2}{\alpha^{j-1}} + \frac{a_1 - ab_1}{\alpha^{j-1}} = \frac{h_{i+1}}{\sqrt{3}\alpha}.$$

Rezultă:

$$g''(i+0) - g''(i-0) = 6(a_i - b_{i-1}) = \frac{2\sqrt{3}}{\alpha} h_{i+1}. \quad (6.13)$$

Acest rezultat are două semnificații importante:

- Pentru orice grup de trei puncte de interpolare P_i, P_{i+1}, P_{i+2} consecutive coliniare, derivata a doua a curbei interpolatoare este continuă în punctul P_i ; prin urmare, curba rezultată este chiar mai netedă decît era de așteptat conform metodei de generare.
- Din expresiile curburii la stînga și la dreapta în punctul i :

$$g''(i-0) = 6 b_{i-1} = 6 \left(-\frac{2h_i}{\sqrt{3}\alpha} - \frac{2h_{i-1}}{\sqrt{3}\alpha^2} - \dots \right)$$

$$g''(i+0) = 6 a_i = 6 \left(\frac{h_{i+1}}{\sqrt{3}\alpha} + b_{i-1} \right) = 6 \left(\frac{h_{i+1}}{\sqrt{3}\alpha} - \frac{2h_i}{\sqrt{3}\alpha} - \frac{2h_{i-1}}{\sqrt{3}\alpha^2} - \dots \right)$$

rezultă că saltul de curbură $\frac{6h_{i+1}}{\sqrt{3}\alpha}$, introdus din cauza aproximărilor impuse de metoda aleasă, este în general de semn contrar curburii la stînga $g''(i-0)$; prin urmare, metoda de interpolare expusă are tendința să reducă curbura în punctele fixate, cu alte cuvinte să „îndrepte” curba interpolatoare.

Formulele simple (6.11) și (6.12) nu ridică probleme de calcul numeric deosebite, mai ales dacă se pot utiliza operații în virgulă mobilă. Dacă nu dispunem decît de aritmetică în virgulă fixă, este necesar să se aleagă aproximări raționale convenabile pentru cantitățile iraționale care apar în formule: $\alpha, \alpha^2, \sqrt{3}$ și $\sqrt{3}\alpha$. La fel ca în cazul general prezentat în secțiunea 4.5, cantitățile $h_i, i \geq 1$ pot fi reprezentate ca numere întregi de 16 biți. Coeficienții $a_i, b_i, i \geq 1$ trebuie reprezentați într-o formă convenabilă de numere raționale, de exemplu cu 16 biți pentru partea întreagă și cu alți 16 biți pentru partea fracționară. De asemenea, trebuie scrise rutine speciale pentru implementarea operațiilor de înmulțire și împărțire care apar în formulele (6.11) și (6.12), care să țină seama de reprezentările particulare ale operanzilor.

Sîntem acum în măsură să prezentăm o procedură de principiu pentru rezolvarea problemei de interpolare puse la începutul acestei secțiuni. Ținînd seama de condițiile problemei, această procedură trebuie să cuprindă următoarele faze:

Pasul 1: Așteaptă definirea punctelor P_0, P_1, P_2, P_3 .

Pasul 2: Calculează câte doi coeficienți a_i, b_i , separat pe fiecare axă, aplicînd de două ori formulele (6.11).

Pasul 3: $i \leftarrow 1$.

Pasul 4: Pornind de la cele două funcții polinomiale de gradul 3 de tipul (6.2), care depind fiecare de o pereche de parametri a_i, b_i , și care constituie împreună o reprezentare parametrică a curbei interpolatoare între punctele P_i și P_{i+1} , trasează efectiv acest arc de curbă, aplicînd procedura descrisă în secțiunea 4.8.

Pasul 5: $i \leftarrow i + 1$.

Pasul 6: Așteaptă definirea punctului P_{i+2} . Dacă acesta nu există, stop.

Pasul 7: Calculează câte doi coeficienți a_i, b_i , separat pe fiecare axă, aplicînd de două ori formule iterative de tipul (6.12).

Pasul 8: Salt la Pasul 4.

Deși numărul de puncte de interpolare poate fi oricît de mare, necesarul de memorie pentru implementarea procedurii de mai sus este limitat la 14 variabile: pentru fiecare axă, avem nevoie de o variabilă pentru parametrul a_i , o variabilă pentru parametrul b_i , 3 variabile pentru coordonatele f_i, f_{i+1} și f_{i+2} și 2 variabile pentru h_i și h_{i+1} .

7. Trasarea curbelor plane definite parametric

În această secțiune ne vom ocupa de problema trasării efective a unui arc de curbă interpolatoare între două puncte consecutive, presupunînd cunoscuți parametrii expresiei analitice a curbei. Problema trasării este principal distinsă de problema interpolării propriu-zise, aceasta din urmă fiind rezolvată odată cu determinarea parametrilor și prin urmare a expresiei analitice a curbei. Cele două probleme se reunesc numai în cazul unei implementări concrete a unei proceduri de interpolare, care să necesite și vizualizarea rezultatelor pe un dispozitiv de afișare grafică.

Dacă evaluăm global timpul de execuție și eficiența unei rutine de interpolare cu vizualizare grafică, observăm că subrutina de trasare este principala consumatoare de timp de execuție. Într-adevăr, timpul de calculare a parametrilor este mai mult sau mai puțin proporțional cu numărul de puncte de interpolare fixate inițial; pe de altă parte, subrutina de trasare trebuie să selecteze toate punctele din mediul de afișare, care aproximează cel mai bine curba ideală, prin evaluări care în general depind de fiecare punct în parte. Între două puncte de interpolare fixate inițial pot exista sute de puncte intermediare pe arcul de curbă care trebuie

generat. În plus, subrutina de trasare trebuie să realizeze operațiile de ieșire propriu-zise asupra mediului de afișare.

În concluzie, eficiența subrutinei de trasare efectivă este de cea mai mare importanță pentru performanțele globale ale unei proceduri de interpolare cu vizualizare grafică.

După aceste considerații preliminare, să reluăm relația (1.6) care definește funcția interpolatoare pe intervalul $[t_i, t_{i+1}]$. Alegînd $t_i = i, d_i = 1$ pentru $i = \overline{0, n-1}$, conform celor discutate în secțiunea 4.3, deducem:

$$g(t) = a_i(i+1-t)^3 + a_{i+1}(t-i)^3 + (f_i - a_i)(i+1-t) + (f_{i+1} - a_{i+1})(t-i), \quad (7.1)$$

$$t \in [i, i+1].$$

În problema specială de interpolare prezentată în secțiunea 4.6 trebuie pornit de la relația (6.2), care de fapt este identică cu (7.1) cu excepția faptului că parametrul a_{i+1} este înlocuit cu b_i . Cu această modificare inițială de notație, toate rezultatele obținute în această secțiune se aplică și problemei speciale de interpolare din secțiunea 4.6.

Efectuăm o translație a originii variabilei independente t la începutul intervalului $[i, i+1]$ și introducem notația:

$$g_i(t) = g(t+i) = a_i(1-t)^3 + a_{i+1}t^3 + (f_i - a_i)(1-t) + (f_{i+1} - a_{i+1})t, \quad (7.2)$$

$$t \in [0, 1].$$

Curba care trebuie trasată între două puncte de interpolare consecutive P_i și P_{i+1} este reprezentată parametric prin două funcții de tipul (7.2), pe care le notăm astfel.

$$g_i^x(t) = g^x(t+i) = a_i^x(1-t)^3 + a_{i+1}^x t^3 + (x_i - a_i^x)(1-t) + (x_{i+1} - a_{i+1}^x)t, \quad (7.3)$$

$$g_i^y(t) = g^y(t+i) = a_i^y(1-t)^3 + a_{i+1}^y t^3 + (y_i - a_i^y)(1-t) + (y_{i+1} - a_{i+1}^y)t,$$

$$t \in [0, 1].$$

În relațiile de mai sus, (x_i, y_i) și (x_{i+1}, y_{i+1}) sînt coordonatele punctelor P_i , respectiv P_{i+1} , iar a_i^x, a_{i+1}^x și a_i^y, a_{i+1}^y sînt parametrii de interpolare relativi la funcțiile $g^x(t)$, respectiv $g^y(t)$ și la intervalul $[i, i+1]$, calculați prin proceduri descrise în secțiunile 4.5 și 4.6.

Să abordăm mai general problema trasării, pentru o curbă a cărei reprezentare parametrică:

$$\begin{aligned} x &= x(t) \\ y &= y(t), \quad t \in [0, 1] \end{aligned} \quad (7.4)$$

conține două funcții continue, care nu sînt neapărat polinoame de gradul 3.

O primă idee pentru dezvoltarea unui algoritm de trasare constă în

a alege o rată constantă Δt de creștere a variabilei independente t și de a selecta în mediul de afișare punctele care corespund cel mai bine adreselor calculate $(x(k\Delta t), y(k\Delta t))$, unde $k = 0, 1, \dots, \frac{1}{\Delta t}$. Principala dificultate constă în alegerea parametrului Δt , deoarece nu se știe de la început câte puncte vor fi efectiv generate. Distanța în linie dreaptă, sau după una din coordonate, între punctele inițial și final este o indicație prea vagă; forma curbei este determinantă în această privință.

Pe de altă parte, însăși reprezentarea parametrică dată (7.4) (care nu este unică!) poate fi defavorabilă, astfel încât orice alegere a unei rate constante de creștere Δt este total nepotrivită: pe unele porțiuni ale curbei, parametrul Δt poate fi prea mic, ceea ce are ca rezultat selectarea de mai multe ori la rând a aceluiași punct, deci ineficiența algoritmului de trasare; pe alte porțiuni ale curbei, același parametru Δt poate fi prea mare, generând puncte consecutive neadiacente în mediul de afișare, care dacă sînt unite prin linii drepte, nu mai aproximează suficient de exact curba dată. De exemplu, reprezentarea parametrică:

$$\begin{aligned}x(t) &= 10t^3 \\y(t) &= 10t^3, \quad t \in [0, 1]\end{aligned}$$

a segmentului de dreaptă dintre punctele $(0, 0)$ și $(10, 10)$, are ca rezultat generarea a două puncte de coordonate întregi $(0, 0)$ și $(1, 1)$ pentru $t \in \left[0, \frac{1}{2}\right]$ și a celorlalte 9 puncte pentru $t \in \left[\frac{1}{2}, 1\right]$.

Prin urmare, orice metodă de generare bazată pe creșterea constantă a variabilei independente nu poate produce algoritmi eficienți în toate cazurile, chiar dacă ne restrîngem la parametrizări cu polinoame de gradul 3. Este necesar să se găsească o metodă bazată pe variația neconstantă a variabilei t , care să urmărească pe cît posibil variația curbei.

Vom expune în continuare o metodă de trasare bazată pe înjumătățirea intervalului de variație a parametrului t . Aceasta reprezintă de fapt o particularizare a ideii generale de rezolvare a unei probleme complicate prin împărțirea ei în două probleme asemănătoare mai simple, care sînt rezolvate pe rînd.

Principiul metodei trasării prin înjumătățirea intervalului este următorul: se calculează mai întîi adresele punctelor de la capetele curbei, corespunzînd extremităților intervalului de variație a parametrului t . Dacă aceste două puncte sînt suficient de apropiate în raport cu rezoluția mediului de afișare, problema este rezolvată prin aprinderea (selectarea) acestor puncte. În caz contrar, intervalul de variație a lui t se împarte în două subintervale de lungimi egale, subintervalul din dreapta se memorează într-o structură de date de tip stivă pentru necesități ulterioare, și se reia problema cu subintervalul din stînga. Înjumătățirea

intervalului continuă pînă cînd lungimea acestuia este suficient de mică pentru ca punctele de pe curbă corespunzătoare extremităților intervalului să fie adiacente în mediul de afișare. În acest caz, se descarcă stiva și se reia procedura cu cel mai recent subinterval memorat în stivă. Trasarea curbei se termină cînd stiva se golește complet.

Corectitudinea procedurii de principiu expusă mai sus se bazează implicit pe faptul că funcțiile de parametrizare $x(t)$ și $y(t)$ din (7.4) sînt continue, prin urmare au proprietatea Darboux; în caz contrar, nu este neapărat necesar ca, între două puncte P_1 și P_2 corespunzînd valorilor $t_1 < t_2$ ale parametrului, toate punctele intermediare (în sensul valorilor absciselor și ordonatelor) să corespundă la valori ale lui t din intervalul $[t_1, t_2]$.

Metoda de trasare prin înjumătățirea intervalului ar putea fi concretizată într-o formă mai concisă cu ajutorul unei proceduri recursive, care să nu facă apel într-un mod explicit la o structură de date de tip stivă. Totuși, preferăm procedura expusă în continuare, nerecursivă, care folosește numai operații simple, ușor de implementat, dacă ținem seama și de faptul că majoritatea microprocesoarelor actuale posedă instrucțiuni pentru manipularea structurilor de tip stivă. Procedura folosește parametrizarea în forma (7.4) pentru curba care trebuie trasată.

Procedură de trasare a unei curbe prin metoda înjumătățirii

Pasul 1: Inițializări: $l \leftarrow 0, r \leftarrow 1, n \leftarrow 0$;

$ixl \leftarrow bint(x(l))$;

$iyl \leftarrow bint(y(l))$.

Pasul 2: $callplot(ixl, iyl)$.

Pasul 3: $ixr \leftarrow bint(x(r))$;

$iyr \leftarrow bint(y(r))$.

Pasul 4: Dacă $|ixr - ixl| \leq 1$ și $|iyr - iyl| \leq 1$, salt la Pasul 6.

Pasul 5: $callpush(r)$;

$r \leftarrow \frac{1}{2}(l + r)$;

$n \leftarrow n + 1$;

salt la Pasul 3.

Pasul 6: $l \leftarrow r, ixl \leftarrow ixr, iyl \leftarrow iyr$.

Pasul 7: Dacă $n = 0$, salt la Pasul 9.

Pasul 8: $callpop(r)$;

$n \leftarrow n - 1$;

salt la Pasul 2.

Pasul 9: *callplot* (*ixl, iyl*) ;

stop.

Procedura folosește 3 variabile de stare l , r (numere reale) și n (întreg) și 4 variabile auxiliare întregi ixl , iyl , ixr , iyr . l și r reprezintă extremitățile stîngă și dreaptă ale intervalului de variație a parametrului t , luat în considerare la un moment dat. n urmărește nivelul curent al stivei.

Subrutina de tip funcție *bint* calculează cea mai bună aproximare în numere întregi a argumentului. Funcția *bint* realizează trecerea de la coordonatele reale de tip $x(t)$ sau $y(t)$ la coordonate întregi, necesare pentru selectarea unui punct în spațiul discret de afișare grafică.

Subrutina *plot* realizează operația fizică de generare a unui punct de coordonate date în mediul discret de afișare. La sfîrșitul acestei secțiuni vom reveni asupra subrutinei *plot*.

Subrutinele *push* și *pop* efectuează încărcarea, respectiv descărcarea stivei cu un element care este specificat ca argument. Se observă că numai extremitatea dreaptă a intervalului este încărcată în stivă; această simplificare față de metoda de principiu este posibilă datorită modului de funcționare a algoritmului: extremitatea stîngă a unui interval care trebuie descărcat din stivă coincide întotdeauna cu extremitatea dreaptă a intervalului analizat în faza anterioară și a cărui lungime s-a redus la minimum (a se vedea Pasul 6).

În Pasul 4 se testează dacă punctele din spațiul de afișare corespunzînd cel mai bine posibil adreselor exacte $(x(l), y(l))$ și $(x(r), y(r))$ sînt suficient de apropiate; în caz negativ, are loc înjumătățirea intervalului $[l, r]$ și încărcarea stivei (în Pasul 5); în caz afirmativ, se descarcă stiva și se continuă procedura cu un interval anterior (pașii 6, 7, 8).

În cazul unui display grafic cu rastru, pentru care spațiul discret de afișare este format din perechile de coordonate întregi ale fiecărui pixel, testul din Pasul 4 are rezultat afirmativ dacă pixelii corespunzînd valorilor l și r ale argumentului t sînt idențici sau adiacenți pe orizontală, verticală sau în diagonală. Prin urmare, curba generată pe un display grafic cu rastru este formată dintr-un șir neîntrerupt de pixeli adiacenți pe orizontală, verticală sau în diagonală.

În cazul unui plotter, testul din Pasul 4 al procedurii are rezultat afirmativ dacă distanța dintre cele două puncte selectate în spațiul grafic este mai mică decît un prag predefinit; ulterior, subrutina *plot* unește cele două puncte printr-o linie dreaptă. Distanța limită din Pasul 4 trebuie aleasă astfel încît o curbă să poată fi aproximată suficient de precis printr-un șir de segmente de dreaptă de lungime mai mică sau egală cu această limită.

În pașii 2 și 9 din procedura de trasare de mai sus, se poate observa că subrutina *plot* acționează totdeauna asupra punctului corespunzînd limitei din stînga a intervalului $[l, r]$ selectat. Acest fapt, împreună cu

alegerea subintervalului din stînga după o operație de înjumătățire a intervalului precedent (subintervalul din dreapta este plasat pe stivă), au ca rezultat generarea punctelor curbei în ordinea crescătoare a variabilei independente t ; mai precis, dacă punctele P_1 și P_2 de pe curbă corespund adreselor $(x(t_1), y(t_1))$, respectiv $(x(t_2), y(t_2))$, și dacă $t_1 < t_2$, atunci algoritmul de mai sus generează punctul P_1 înaintea lui P_2 .

Acest efect, care nu era singura posibilitate, a fost urmărit în mod special, deoarece există situații cînd ordinea generării punctelor curbei are importanță. De exemplu, dacă trasarea curbei trebuie să se efectueze nu cu linie continuă, ci cu linie întreruptă corespunzînd unui pattern (configurație liniară de biți) predefinit, atunci algoritmul de mai sus permite calcularea punctelor succesive ale curbei în paralel cu parcurgerea configurației binare a pattern-ului, care se repetă în mod circular; punctul curent de pe curbă este efectiv selectat în mediul de afișare numai dacă bitul corespunzător din pattern este 1.

În legătură cu generarea curbei conform unui pattern dat, există o situație specială care trebuie luată în considerare: trasarea cu pattern descrisă mai sus nu va fi complet reușită dacă algoritmul de trasare generează de două ori la rînd același punct pe curbă. Procedura de mai sus poate ajunge efectiv în această situație, datorită aproximărilor impuse de modul de adresare grafică sau din cauza unei parametrizări defavorabile a curbei. Există două moduri de protecție contra generării de două ori la rînd a aceluiași punct:

- prin complicarea algoritmului principal, pornind de la ideea de a separa în Pasul 4 cazul cînd cele două puncte din mediul de afișare (corespunzînd valorilor l și r ale argumentului t) sînt identice, de cazul în care cele două puncte sînt numai vecine;
- prin trecerea acestei probleme în sarcina subrutinei *plot*. În acest caz, problema se rezolvă simplu prin introducerea a două variabile locale în subrutina *plot*, care să păstreze în permanență coordonatele x, y din spațiul de afișare pentru ultimul punct generat, și care să inhibe generarea efectivă a unui punct dacă coordonatele acestuia coincid cu cele ale punctului generat anterior.

4.8. Algoritm de trasare eficientă

Să examinăm acum mai atent elementele care influențează eficiența în execuție a algoritmului de trasare descris în secțiunea 4.7. Subrutinele *plot*, *push* și *pop* sînt mai mult sau mai puțin „apeluri de sistem”, determinate de configurația hardware, prin urmare asupra lor nu se poate acționa. Celelalte operații efectuate de algoritm, inclusiv aproximările la

Întregul cel mai apropiat executate de rutina *bin*, sînt foarte simple, cu excepția calculării efective a valorilor funcțiilor $x(t)$ și $y(t)$ efectuate în pași 1 și 3 ai algoritmului. Într-adevăr, evaluarea funcțiilor de parametrizare $x(t)$ și $y(t)$ poate implica operații de înmulțire, împărțire, extrageri de rădăcini pătrate, etc., mai dificil de implementat și/sau mai lente pentru procesoarele dispozitivelor periferice actuale. Eficiența operațiilor de evaluare a acestor funcții în Pasul 3 este de fapt determinantă pentru performanțele practice ale algoritmului, adică pentru viteza de trasare.

Dacă problema trasării este pusă în legătură cu un mecanism de interpolare a curbelor plane, și dacă interpolarea se bazează pe metoda funcțiilor spline cubice pe porțiuni, funcțiile de parametrizare a curbei interpolatoare au o formă particulară — sînt polinoame de gradul 3 (sau mai mic). Prin urmare, să examinăm mai atent posibilitățile de evaluare a funcțiilor de parametrizare de acest tip:

$$\begin{aligned} x(t) &= A_x t^3 + B_x t^2 + C_x t + D_x \\ y(t) &= A_y t^3 + B_y t^2 + C_y t + D_y, \quad t \in [0, 1]. \end{aligned} \quad (8.1)$$

Calea directă cea mai eficientă de evaluare a acestor funcții pare a fi cea bazată pe schema lui Horner:

$$x(t) = ((A_x t + B_x) t + C_x) t + D_x \quad (8.2)$$

și analog pentru $y(t)$, ceea ce conduce la efectuarea a 6 înmulțiri și 6 adunări la fiecare trecere prin Pasul 3 al algoritmului de trasare.

Căutînd posibilitați de a reduce cît mai mult numărul de înmulțiri necesare, ajungem la următoarele formule, valabile pentru orice polinom de gradul 3 (sau mai mic) cu notația (8.1):

$$\begin{aligned} x\left(t - \frac{\Delta}{2}\right) &= \frac{1}{8} [3x(t - \Delta) + 6x(t) - x(t + \Delta) + 3A_x \Delta^3] \\ x\left(t + \frac{\Delta}{2}\right) &= \frac{1}{8} [3x(t + \Delta) + 6x(t) - x(t - \Delta) - 3A_x \Delta^3], \quad \forall t, \Delta \end{aligned} \quad (8.3)$$

și analoagele pentru $y(t)$. Dacă, de exemplu, cunoaștem funcția $x(t)$ în punctele $t - \Delta$, t și $t + \Delta$, atunci pentru calcularea valorii $x\left(t - \frac{\Delta}{2}\right)$ nu sînt necesare înmulțiri decît pentru termenul $3A_x \Delta^3$, deoarece înmulțirea cu 6 se poate efectua prin două deplasări binare la stînga și o adunare, împărțirea cu 8 prin trei deplasări la dreapta, etc. Dacă vom pune în legătură aceste proprietăți ale polinoamelor de gradul 3 cu metoda de trasare prin înjumătățirea intervalului, vom ajunge la o concluzie pe cît de surprinzătoare, pe atît de încurajatoare pentru o implementare practică: este posibilă dezvoltarea unui algoritm de trasare care nu necesită nici o înmulțire, și care evaluează funcțiile în toate punctele necesare numai prin adunări, scăderi și deplasări binare!

În algoritmul de trasare prezentat în secțiunea 4.7, lăsând la o parte variabila n de control al stivei, am utilizat două variabile de stare l și r pentru delimitarea subintervalului din domeniul parametrului t pe care erau evaluate funcțiile $x(t)$ și $y(t)$. Pentru dezvoltarea unui algoritm de trasare mai eficient, este necesar să alegem un alt set de variabile de stare, eventual mai bogat, care să permită punerea în valoare a proprietăților exprimate prin formulele (8.3). Vom renunța la variabilele de stare din domeniul parametrului t , care în fond nu sînt utilizate direct, și vom alege variabile de stare din domeniul de valori ale funcțiilor $x(t)$ și $y(t)$. Concret, vom utiliza 8 variabile de stare (numere reale), cîte 4 pentru fiecare axă, precizate astfel:

$$\begin{aligned} (xl, xm, xr, xk) &= \left(x(l), x\left(\frac{l+r}{2}\right), x(r), 3A_x\left(\frac{r-l}{2}\right)^3 \right) \\ (yl, ym, yr, yk) &= \left(y(l), y\left(\frac{l+r}{2}\right), y(r), 3A_y\left(\frac{r-l}{2}\right)^3 \right) \end{aligned} \quad (8.4)$$

unde l și r au aceeași semnificație ca în algoritmul din secțiunea 4.7, dar nu mai apar în mod explicit. După cum se vede, am introdus multă redundanță alegînd atît de multe variabile de stare, dar această alegere simplifică la maximum operațiile aritmetice pe care trebuie să le efectueze algoritmul. Astfel, în loc de a calcula $x(l)$, $x(r)$, $y(l)$ și $y(r)$ se vor utiliza direct variabilele xl , xr , yl și respectiv yr .

La înjumătățirea intervalului $[l, r]$, operație care are loc în Pasul 5, se vor calcula două noi seturi de variabile, unul referitor la subintervalul din stînga $\left[l, \frac{l+r}{2}\right]$, cu care continuă execuția algoritmului, și altul referitor la subintervalul din dreapta $\left[\frac{l+r}{2}, r\right]$, care este plasat pe stivă pentru reluare ulterioară. Variabilele referitoare la subintervalul $\left[l, \frac{l+r}{2}\right]$ se calculează astfel:

$$\begin{aligned} (xl', xm', xr', xk') &\leftarrow \left(x(l), x\left(\frac{3l+r}{4}\right), x\left(\frac{l+r}{2}\right), 3A_x\left(\frac{r-l}{4}\right)^3 \right) = \\ &= \left(xl, xml, xm, \frac{xk}{8} \right) \end{aligned}$$

unde xl , xm , xk sînt variabile referitoare la intervalul inițial $[l, r]$, iar xml (notație folosită în algoritm) se calculează în conformitate cu prima formulă (8.3):

$$xml = x\left(\frac{3l+r}{4}\right) = \frac{1}{8} \left[3x(l) + 6x\left(\frac{l+r}{2}\right) - x(r) + 3A_x\left(\frac{r-l}{2}\right)^3 \right] =$$

$$= \frac{1}{8} (3xl + 6xm - xr + xk) .$$

Variabilele referitoare la subintervalul din dreapta $\left[\frac{l+r}{2}, r \right]$ se calculează astfel:

$$\begin{aligned} (xl'', xm'', xr'', xk'') &\leftarrow \left(x\left(\frac{l+r}{2}\right), x\left(\frac{l+3r}{4}\right), x(r), 3A_x\left(\frac{r-l}{4}\right)^3 \right) = \\ &= \left(xm, xmr, xr, \frac{xk}{8} \right) \end{aligned}$$

unde xm, xr, xk sînt variabile referitoare la intervalul inițial $[l, r]$, iar xmr (notație folosită în algoritm) se calculează în conformitate cu a doua formulă (8.3):

$$\begin{aligned} xmr &= x\left(\frac{l+3r}{4}\right) = \frac{1}{8} \left[3x(r) + 6x\left(\frac{l+r}{2}\right) - x(l) - 3A_x\left(\frac{r-l}{2}\right)^3 \right] = \\ &= \frac{1}{8} (3xr + 6xm - xl - xk) . \end{aligned}$$

Relații absolut analoage au loc pentru transformările variabilelor (yl, ym, yr, yk) .

Celelalte variabile utilizate, precum și subrutinele *bint*, *plot*, *push* și *pop*, au aceleași semnificații ca în cazul algoritmului general din secțiunea 4.7.

După aceste precizări, putem trece la descrierea variantei îmbunătățite a algoritmului de trasare a unei curbe prin metoda înjumătățirii intervalului. Procedura prezentată pornește de la reprezentarea parametrică (8.1). Algoritmul rezultat este mai amplu decât cel prezentat în secțiunea 4.7, dar nu și mai complex; acest fapt este pus în evidență prin numerotarea identică a fazelor de execuție. În mod neîndoiește, algoritmul care urmează este mai eficient, deoarece, cum am arătat anterior, nu utilizează decât operații aritmetice simple de adunare, scădere și deplasare binară.

Procedură eficientă de trasare fără înmulțiri și împărțiri

Pasul 1: Inițializări: $xl \leftarrow x(0) = D_x$;

$$xm \leftarrow x\left(\frac{1}{2}\right) = \frac{A_x}{8} + \frac{B_x}{4} + \frac{C_x}{2} + D_x ;$$

$$xr \leftarrow x(1) = A_x + B_x + C_x + D_x ;$$

$$xk \leftarrow \frac{3A_x}{8} ;$$

$$yl \leftarrow y(0) = D_y ;$$

$$\begin{aligned} ym &\leftarrow y \left(\frac{1}{2} \right) = \frac{A_y}{8} + \frac{B_y}{4} + \frac{C_y}{2} + D_y; \\ yr &\leftarrow y(1) = A_y + B_y + C_y + D_y; \\ yk &\leftarrow \frac{3A_y}{8}; \\ n &\leftarrow 0; \\ ixl &\leftarrow \text{bint}(xl); \\ iyl &\leftarrow \text{bint}(yl). \end{aligned}$$

Pasul 2: $\text{callplot}(ixl, iyl)$.

Pasul 3: $ixr \leftarrow \text{bint}(xr);$
 $iyr \leftarrow \text{bint}(yr).$

Pasul 4: Dacă $|ixr - ixl| \leq 1$ și $|iyr - iyl| \leq 1$, salt la Pasul 6.

Pasul 5: $xml \leftarrow \frac{1}{8}(3xl + 6xm - xr + xk);$

$$xmr \leftarrow \frac{1}{8}(3xr + 6xm - xl - xk);$$

$$xk \leftarrow \frac{1}{8}xk;$$

$\text{callpush}(xr);$

$\text{callpush}(xmr);$

$\text{callpush}(xk);$

$xr \leftarrow xm;$

$xm \leftarrow xml;$

$$yml \leftarrow \frac{1}{8}(3yl + 6ym - yr + yk);$$

$$ymr \leftarrow \frac{1}{8}(3yr + 6ym - yl - yk);$$

$$yk \leftarrow \frac{1}{8}yk;$$

$\text{callpush}(yr);$

$\text{callpush}(ymr);$

$\text{callpush}(yk);$

$yr \leftarrow ym;$

$ym \leftarrow yml;$

$n \leftarrow n + 1;$

salt la Pasul 3.

Pasul 6: $xl \leftarrow xr, ixl \leftarrow ixr;$

$yl \leftarrow yr, iyl \leftarrow iyr.$

Pasul 7: Dacă $n = 0$, salt la Pasul 9.

Pasul 8: $\text{callpop}(yk);$

$\text{callpop}(ym);$

```

callpop ( yr );
callpop ( xk );
callpop ( xm );
callpop ( xr );
n ← n - 1;
salt la Pasul 2.

```

Pasul 9: *callplot* (*ixl*, *iyl*);
stop.

Pentru salvarea și refacerea unui subinterval în și din stivă sînt necesare în acest caz cîte 6 operații de *push*, respectiv *pop* pentru variabilele de stare. După cum s-a arătat în secțiunea 4.7, nu este necesară salvarea variabilelor *xl*, *yl* referitoare la marginea stîngă a subintervalului.

Procedura prezentată mai sus poate fi utilizată aproape fără modificări pentru realizarea modului de vizualizare în cadrul unui program general de interpolare cu vizualizare a curbelor plane. Toate modificările necesare se referă la inițializările variabilelor din Pasul 1, care în algoritmul prezentat folosesc reprezentarea parametrică generală (8.1). Particularizînd pentru expresiile (7.3), care reprezintă parametrizarea unui arc de curbă între două puncte de interpolare consecutive P_i și P_{i+1} , inițializările variabilelor procedurii se rescriu astfel:

Pasul 1: Inițializări: $xl \leftarrow g_i^x(0) = x_i$;

$$xm \leftarrow g_i^x\left(\frac{1}{2}\right) = \frac{1}{2}(x_i + x_{i+1}) - \frac{3}{8}(a_i^x + a_{i+1}^x);$$

$$xr \leftarrow g_i^x(1) = x_{i+1};$$

$$xk \leftarrow \frac{3}{8}(a_{i+1}^x - a_i^x);$$

$$yl \leftarrow g_i^y(0) = y_i;$$

$$ym \leftarrow g_i^y\left(\frac{1}{2}\right) = \frac{1}{2}(y_i + y_{i+1}) - \frac{3}{8}(a_i^y + a_{i+1}^y);$$

$$yr \leftarrow g_i^y(1) = y_{i+1};$$

$$yk \leftarrow \frac{3}{8}(a_{i+1}^y - a_i^y);$$

$$n \leftarrow 0;$$

$$ixl \leftarrow bint(xl);$$

$$iyl \leftarrow bint(yl).$$

În relațiile de mai sus, (x_i, y_i) și (x_{i+1}, y_{i+1}) sînt coordonatele punctelor P_i , respectiv P_{i+1} , iar a_i^x , a_{i+1}^x , a_i^y , a_{i+1}^y sînt parametrii arcului de curbă interpolatoare între P_i și P_{i+1} .

Se observă că și în faza de inițializare sînt suficiente numai operații de adunare, scădere și deplasare binară pentru efectuarea calculului.

După cum s-a arătat în secțiunile 4.5 și 4.6, dacă nu se pot folosi operații în virgulă mobilă, parametri de interpolare pot fi reprezentați cu suficientă precizie în virgulă fixă, sub formă de numere raționale cu 16 biți la partea întreagă și 16 biți la partea fracționară. Deoarece coordonatele $x_i, y_i, x_{i+1}, y_{i+1}$ sînt numere întregi, rezultă că variabilele algoritmului de trasare $x_l, x_m, x_r, x_k, y_l, y_m, y_r, y_k$ pot fi reprezentate ca numere raționale de 32 biți, cu 16 biți pentru partea întreagă și 16 biți pentru partea fracționară.

O ultimă problemă, importantă, care trebuie analizată pentru a asigura funcționarea corectă a algoritmului de trasare, mai ales în varianta îmbunătățită de mai sus, este valoarea maximă n_{\max} la care poate ajunge indicatorul stivei n . Această valoare permite alegerea corectă a dimensiunii zonei de memorie care trebuie rezervată pentru stivă în procedura de trasare. Valoarea n_{\max} depinde într-un mod complicat de funcțiile de parametrizare $x(t)$ și $y(t)$.

Putem ajunge la o estimare superioară a lui n_{\max} în modul următor: din algoritmul de trasare rezultă că are loc o înjumătățire a intervalului curent $[l, r]$, și implicit o incrementare a indicatorului stivei n , de fiecare dată cînd:

$$\max \{ |x(r)-x(l)|, |y(r)-y(l)| \} \geq 2 \quad (8.5)$$

(fără să mai ținem seama de operațiile de rotunjire la întregi). Dintr-o teoremă elementară de analiză deducem:

$$|x(r)-x(l)| \leq (r-l) \sup_{t \in [l,r]} |x'(t)| \leq (r-l) \sup_{t \in [0,1]} |x'(t)| = (r-l) m_x \quad (8.6)$$

$$|y(r)-y(l)| \leq (r-l) \sup_{t \in [l,r]} |y'(t)| \leq (r-l) \sup_{t \in [0,1]} |y'(t)| = (r-l) m_y$$

unde am notat $m_x = \sup_{t \in [0,1]} |x'(t)|$, $m_y = \sup_{t \in [0,1]} |y'(t)|$. Să considerăm acum funcțiile de parametrizare:

$$x_1(t) = m_x t \quad (8.7)$$

$$y_1(t) = m_y t, \quad t \in [0, 1].$$

Din inegalitățile (8.6) rezultă:

$$\max \{ |x(r)-x(l)|, |y(r)-y(l)| \} \leq \max \{ |x_1(r)-x_1(l)|, |y_1(r)-y_1(l)| \}$$

pentru orice interval $[l, r] \subset [0, 1]$. De aici deducem:

$$n_{\max} \leq n_{1 \max} \quad (8.8)$$

unde $n_{1 \max}$ este valoarea maximă atinsă de indicatorul stivei atunci cînd

algoritmul de trasare se aplică pentru curba $(x_1(t), y_1(t))$.

Pe de altă parte, se vede ușor că indicatorul n_{\max} este în dependență directă cu lățimea minimă Δ_{\min} a intervalelor $[l, r]$ analizate de algoritm, și anume:

$$\Delta_{\min} = \frac{1}{2^{n_{\max}}} \quad (8.9)$$

dacă ținem seama de faptul că inițial $[l, r] = [0, 1]$, $\Delta = 1$ și $n = 0$.

În cazul curbei cu parametrizarea (8.7), cantitatea $\Delta_{1 \min}$ este ușor de dedus; pentru această curbă, relația (8.5) se scrie:

$$(r-l) \max \{m_x, m_y\} \geq 2. \quad (8.10)$$

Atât timp cît pentru intervalul curent $[l, r]$ este îndeplinită condiția (8.10), continuă procesul de înjumătățire a intervalului; lățimea minimă $\Delta_{1 \min}$ este atinsă cînd:

$$\frac{1}{2} \leq \Delta_{1 \min} \max \{m_x, m_y\} \leq 1. \quad (8.11)$$

Din relațiile (8.8), (8.9) și (8.11) rezultă:

$$n_{\max} \leq n_{1 \max} = \log_2 \frac{1}{\Delta_{1 \min}} \leq 1 + \log_2 \max \{m_x, m_y\} \quad (8.12)$$

unde $m_x = \sup_{t \in [0,1]} |x'(t)|$, $m_y = \sup_{t \in [0,1]} |y'(t)|$. Aceste relații furnizează o margine superioară pentru nivelul stivei n în cazul trasării unei curbe de forma generală (7.4). În cazul particular al trasării după interpolare, putem adînci analiza datorită formei particulare (7.3) a funcțiilor de parametrizare. În acest caz avem:

$$\begin{aligned} x'(t) &= -3a_i^x(1-t)^2 + 3a_{i+1}^x t^2 - (x_i - a_i^x) + (x_{i+1} - a_{i+1}^x) = \\ &= [1-3(1-t)^2] a_i^x + (3t^2-1) a_{i+1}^x + (x_{i+1} - x_i) \end{aligned}$$

$$\sup_{t \in [0,1]} |x'(t)| \leq 2|a_i^x| + 2|a_{i+1}^x| + |x_{i+1} - x_i|$$

și relația analoagă pentru $y'(t)$. Deoarece x_i, x_{i+1} sînt coordonate întregi în spațiul de afișare, putem considera $|x_{i+1} - x_i| \leq 1000$. De asemenea, din analizele prezentate în secțiunile 4.5 și 4.6 rezultă că parametrii de interpolare a_i, a_{i+1} sînt cantități de ordinul de mărime al diferențelor $h_i = x_{i-1} - 2x_i + x_{i+1}$, deci putem considera $|a_{i+1}^x| \leq 1000, |a_i^x| \leq 1000$ (în mod normal acești parametri sînt cel mult de ordinul zecilor). Avem deci estimarea:

$$\sup_{t \in [0,1]} |x'(t)| \leq 5000$$

și relația anăloagă pentru $y'(t)$. În definitiv, din (8.12) obținem:

$$n_{\max} \leq 1 + \log_2 5000 < 15.$$

Acest rezultat dovedește că în procedura eficientă de trasare a unui arc de curbă interpolatoare, descrisă mai înainte în această secțiune, este suficient să se rezerve pentru stivă o zonă de memorie de $6 \times 15 = 90$ cantități de 32 biți fiecare.

4.9. Bibliografie

-
- [1] T. Pavlidis
Algorithms for Graphics and Image Processing, Springer Verlag, 1982
 - [2] W.M. Newman, R.F. Sproull
Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill, 1979
 - [3] J.D. Foley, A. Van Dam
Fundamentals of Interactive Computer Graphics, Addison Wesley, 1983
 - [4] K.S. Fu, T.L. Kunii
Picture Engineering, Springer Verlag, 1982
 - [5] G.I. Marciuk
Metode de analiză numerică, Ed. Academiei RSR, București 1983
 - [6] J.E. Bresenham, R.A. Earnshaw, A.R. Forrest, R.J. Lansdown, M.L.V. Pitteway
Theoretical Foundations of Computer Graphics and CAD, NATO ASI Series, Springer Verlag, 1988

- Obiectivul cărții este acela de a reuni aspectele de implementare și de aplicații pentru grafica pe calculator în limbajele moderne de programare PASCAL și C, unele dintre subiectele abordate fiind în premieră la noi în țară.
- Cartea se adresează studenților, chiar elevilor, cercetătorilor, cadrelor didactice și tuturor utilizatorilor calculatoarelor electronice din domeniul cercetării și proiectării asistate de calculator.
- Utilitatea cărții rezultă din faptul că subiectele tratate sînt rezultatul colaborării dintre specialiști ce lucrează în învățămînt, cercetare și în domeniul elaborării de software, și sînt susținute de multe programe executate pe calculator, împreună cu imaginile grafice corespunzătoare.



- **VOLUMUL I**, intitulat „IMPLEMENTARE”, abordează particularitățile de implementare a aspectelor grafice atît pe minicalculatoare (compatibile PDP), cît și pe microcalculatoare (compatibile IBM-PC).
- **VOLUMUL II**, intitulat „APLICAȚII”, abordează aplicații de grafică pe calculator în domeniile: teoria fractalilor, teoria curbilor și suprafețelor, geometria de tip „TURTLE”, trasarea curbilor pe dispozitivele de afișare grafică.